KHAZAR UNIVERSITY

School: Architecture, engineering and applied sciences.
Department: Computer Science and Management
Major: Computer Science

MS Thesis

Title: "Review of the operations over the digital certificates in various cases"

Student: Nargiz Farhad Gurbanova

Supervisor: PhD. Sakhavat Talibov

Baku – 2009

Content

Appendix

Abstract

“Review of the operations over the digital certificates in various cases”

This thesis will study the problems of certificates their vulnerability, and alternative solutions. The proposed research plan includes experiments on studying the security and performance aspects of each of the alternative solutions. Appendix contains the programming part.

In Chapter 1 is described some theoretical aspects and brief overview about HTTP, HTTPS, their difference, digital certificates, and etc. Chapter 2 summarizes the state HTTP and HTTPS connection, SSL properties, Server Authentication. In third chapter is described the operation over the digital certificates as untangling digital certificates, obtaining of certificates, installing the certificates, the graphics, and the designing and implementation of the program with easy structure which allow to do operations over the certificates in Windows operation system. The purpose of this thesis is to show the operation over the digital certification connecting the problems related to the changing of the wrong certificates and designing and implementation of the program.

The following are the three most common security risks CGI applications and SSI pages create: Information leaks, depleting system resources, access to potentially dangerous system commands/applications.The intent was to create a model that improved the way revocation is handled in terms of: Revocation time, Security, Scalability. The proposed solution is based on the X.509 framework. The reason why I choose this framework was: It uses a centralized certificate authority and it has a very well defined structure.

Referat

“Rəqəmsal sertifikatlar üzərində müxtəlif hallarda əməliyyatların təhlili”

Diplom işində sertifikatların zəif tərəfləri və alternativ həllər təhlil olunur. Təklif olunan tədqiqat planı eksperiment və yerinə yetirilən aspektləri özündə cəmləşdirir.
Birinci fəsildə ümumi nəzəri aspektlər, Http və Https haqqında və onların fərqi haqqında, sertifikatlar haqqında qısa məlumat verilib. Ikinci fəsildə isə Http və Https əlaqəsi, SSL xüsusiyyətləri, serverdə ausentifikasiyası təsvir olunur. Üçüncü fəsildə sertifikatlar üzərində müxtəlif hallarda əməliyyataŕın təhlili, Windows əməliyyat sistemində sertifikatlar üzərində əməliyyatların asan strukturalı proqramın yerinə yetirməsini, , hansı ki Windows əməliyyatı sistemində sertifikatlar üzərində əməliyyatlar keçirməyə üçün imkan verir.
Aşağıda göstərilən üç ümumi təhlükəsizlik riskləri CGI tətbiqləri və SSL yaradır: İnformasiya sızmaları: Hakerə istənilən sistem informasiyası serverinizdən gönürməyə imkan verir. Sistem resurslarının tükənməsi.
Məqsəd ele bir model yaratmaq idi ki, ləğv etmə vaxtını, təhlükəsizliyi, əhatəliliyi təkmilləşdirir.
Təklif edilmiş həll X.509 struktura əsaslanır. Səbəbi isə onun mərkəzləşdirmiş sertifikatlar avtorizasiyası istifadə edir və o çox yaxşı struktura malikdir.
.Əlavədə isə təklif olunan proqramın proqramlaşdirma hissəsi daxildir.

Chapter 1.

Introduction

1.2 Hypertext Transfer Protocol

In this chapter is described some theoretical aspects.

Hypertext Transfer Protocol (HTTP) is a communications protocol used to transfer or convey information on the World Wide Web. Its original purpose was to provide a way to publish and retrieve HTML hypertext pages. HTTP is a request/response protocol between clients and servers. The originating client, such as a web browser, spider, or other end-user tool, is referred to as the user agent. The destination server, which stores or creates resources such as HTML files and images, is called the origin server. In between the user agent and origin server may be several intermediaries, such as proxies, gateways, and tunnels. It is useful to remember that HTTP does not need TCP/IP. Indeed HTTP can be "implemented on top of any other protocol on the Internet, or on other networks. HTTP only presumes a reliable transport; any protocol that provides such guarantees can be used."

There are currently two methods of establishing a secure HTTP connection: the https URI scheme and the HTTP 1.1 Upgrade header. The https URI scheme has been deprecated by RFC 2817, which introduced the Upgrade header; however, as browser support for the Upgrade header is nearly non-existent, the https URI scheme is still the dominant method of establishing a secure HTTP connection.

https URI Scheme

https: is a URI scheme syntactically identical to the http: scheme used for normal HTTP connections, but which signals the browser to use an added encryption layer of SSL/TLS to protect the traffic. SSL is especially suited for HTTP since it can provide some protection even if only one side of the communication is authenticated. In the case of HTTP transactions over the Internet, typically only the server side is authenticated

HTTP 1.1 Upgrade header

HTTP 1.1 introduced support for the Upgrade header. In the exchange, the client begins by making a clear-text request, which is later upgraded to TLS. Either the client or the server may request (or demand) that the connection be upgraded. The most common usage is a clear-text request by the client followed by a server demand to upgrade the connection, which looks like this:

Client:

GET /encrypted-area HTTP/1.1
Host: www.example.com

Server:

HTTP/1.1 426 Upgrade Required
Upgrade: TLS/1.0, HTTP/1.1
Connection: Upgrade

The server returns a 426 status-code because 400 level codes indicate a client failure (see List of HTTP status codes), which correctly alerts legacy clients that the failure was client-related.

The benefits of using this method for establishing a secure connection are:

Sample

Below is a sample conversation between an HTTP client and an HTTP server running on www.example.com, port 80.

Client request (followed by a blank line, so that request ends with a double newline, each in the form of a carriage return followed by a line feed):

GET /index.html HTTP/1.1
Host: www.example.com

The "Host" header distinguishes between various DNS names sharing a single IP address, allowing name-based virtual hosting. While optional in HTTP/1.0, it is mandatory in HTTP/1.1.

Server response (followed by a blank line and text of the requested page):

HTTP 200 OK
Date: Mon, 15 May 2009 22:38:34 GMT
Server: Apache/1.3.27 (Unix) (Red-Hat/Linux)
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
Etag: "3f80f-1b6-3e1cb03b"
Accept-Ranges: bytes
Content-Length: 438

Connection: close
Content-Type: text/html; charset=UTF-8

Reading and writing data using HTTP

1. Obtain the HttpConnection Object—Before data can be read or written from and to the server, a HttpConnection object is needed. Connector class of javax.microedition.io package provides three ways to obtain the HttpConnection Object—

• HttpConnection httpConn = (HttpConnection)Connector.open(String url);
• HttpConnection httpConn = (HttpConnection)Connector.open(String url, int mode);
• HttpConnection httpConn = (HttpConnection)Connector.open(String url, int mode, boolean timeout);

In the above methods—

• url—is the url of the server. For HttpConnection it will be of the format—http://www.site.com.
• mode—specifies one of the READ, READ_WRITE or WRITE modes. If no mode is specified, the default mode is READ_WRITE, i.e. both read and write operations can be performed .
• timeout—a flag to indicate if the caller wants to see the exceptions on time out. If this flag is true and if connection cannot be made after 60 seconds, the open() will throw IOException. If no flag is specified, the default value is true.

2. Reading from the server—The HttpConnection object so obtained can be used to read data from the server by obtaining the input stream or data input stream. The input stream can be obtained by using the openInputStream() of the obtained HttpConnection object, whereas the data input stream can be obtained by using the openDataInputStream() of HttpConnection object as shown below—

• InputStream is = httpConn.openInputStream();
• DataInputStream dis = htppConn.openDataInputStream();

Data can be read using the read method of the InputStream and DataInputStream class. It should be noted that if no data can be read from the server within 40 seconds, the read method will throw IOException.

3. Writing to server— In order to write or post some data to a server, an output stream or data output stream is required. The request method—POST should be specified by using setRequestMethod() of

HttpConnection, before writing data to server. It is also recommended to send the Content-Length header in the request with the exact length of data to be read. After setting the headers, the output stream can be obtained by using the openOutputStream method of the ttpConnection, whereas the data output stream can be obtained by using the openDataOutputStream() of HttpConnection as shown below—

• OutputStream os = httpConn.openOutputStream();

• DataOutputStream dos = httpConn.openDataOutputStream();

Data can then be written using the write method of the OutputStream or DataOutputStream. It should be noted that if no data can be written to the server within 60 seconds, the read method will throw OException.

1.3 **Https**

https is a URI scheme used to indicate a secure HTTP connection. It is syntactically identical to the http:// scheme normally used for accessing resources using HTTP. Using an https: URL indicates that HTTP is to be used, but with a different default TCP port (443) and an additional encryption/authentication layer between the HTTP and TCP. This system was designed by Netscape Communications Corporation to provide authentication and encrypted communication and is widely used on the World Wide Web for security-sensitive communication such as payment transactions and corporate logons.

Https is not a separate protocol, but refers to the combination of a normal HTTP interaction over an encrypted Secure Sockets Layer (SSL) or Transport Layer Security (TLS) transport mechanism. This ensures reasonable protection from eavesdroppers and man-in-the-middle attacks, provided it is properly implemented and the top level certification authorities do their job.

The default TCP port of an https: URL is 443 (for unsecured HTTP, the default is 80).

To prepare a web-server for accepting https connections the administrator must create a public key certificate for the web-server. These certificates can be created for Unix based servers with tools such as OpenSSL's ssl-ca or SuSE's gensslcert. This certificate must be signed by a certificate authority of one form or another, who certifies that the certificate holder is who they say they are. Web browsers are generally distributed with the signing certificates of major certificate authorities, so that they can verify certificates signed by them.

Organizations may also run their own certificate authority, particularly if they are responsible for setting up browsers to access their own sites (for example, sites on a company intranet), as they can trivially add their own signing certificate to the defaults shipped with the browser.

Some sites use self signed certificates. Using these provides protection against pure eavesdropping but unless the certificate is verified by some other method (for example, phoning the certificate owner to verify its checksum) and that other method is secure, there is a risk of a man-in-the-middle attack.

The system can also be used for client authentication, in order to restrict access to a Web server to only authorized users. For this, typically the site administrator creates certificates for each user which are loaded into their browser, although certificates signed by any certificate authority the server trusts should work. These normally contain the name and e-mail of the authorized user, and are automatically checked by the server on each reconnect to verify the user's identity, potentially without ever entering a password.

HTTPS configuration

The httpsConfig protocol configuration parameter is used for requests using the https protocol to specify the certificate and trust for the outbound HTTPS connection.

The value of httpsConfig names an outbound HTTPS configuration that has been defined as /config/connection/https/ configName. If no httpsConfig value was associated with the request using the mechanisms described in Configuring protocols the supplied configuration, defaultConfig, will be used.

The behavior of the defaultConfig configuration depends on whether the application includes the zero.core.webtools dependency:

- If zero.core.webtools is included, a development and test configuration is assumed:
- Outbound HTTPS connections using defaultConfig will not perform any checks to validate the identity of the remote server.
- If zero.core.webtools is not included, a simple trust configuration is assumed:
- Outbound HTTPS connections using defaultConfig will check that the remote sever presents a valid certificate signed by one of a selection of well-known certificate authorities.

The HTTPS connection configurations defined under /config/connection/https/ configName can contain the following properties:

Property	Description	Notes
trustStore	File name of trust store to be used to validate server identity.	Required.
trustStorePassword	Trust store password. (XOR strings are supported)	Required.
trustStoreType	Trust store type.	Required.
keyStore	File name of key store containing client certificate.	Optional, default is no client certificate.
keyStorePassword	Key store password. (XOR strings are supported)	Required if keyStore is specified.
keyStoreType	Key store type.	Required if keyStore is specified.
disableTrustVerification	If true, the outbound connection will not perform any checks to validate the identity of the remote server.	Optional, default value is false

Limitations

The level of protection depends on the correctness of the implementation by the web browser and the server software and the actual cryptographic algorithms supported.

A common misconception among credit card users on the Web is that https: fully protects their card number from thieves. In reality, an encrypted connection to the Web server only protects the credit card number in transit between the user's computer and the server itself. It doesn't guarantee that the server itself is secure, or even that it hasn't already been compromised by an attacker.

Attacks on the Web sites that store customer data are both easier and more common than attempts to intercept data in transit. Merchant sites are supposed to immediately forward incoming transactions to a payment gateway and retain only a transaction number, but they often save card numbers in a database. It is that server and database that is usually attacked and compromised by unauthorized users.

Because SSL operates below http and has no knowledge of the higher level protocol, SSL servers can only present one certificate for a particular IP/port combination. This means that in most cases it is not feasible to use name-based virtual hosting with HTTPS.

Reading and writing data using HttpsConnection

1. Reading and writing data using HttpsConnection is very similar to reading data over the HttpConnection (as discussed previously in steps 1 to 3), except for the way the HttpsConnection object is obtained. The HttpsConnection object can be obtained by replacing the HttpConnection in step 1 of previous section with the HttpsConnection. So the HttpsConnection object can be obtained from one of the following open methods of Connector class—

• HttpsConnection httpsConn = (HttpsConnection)Connector.open(String url);
• HttpsConnection httpsConn = (HttpsConnection)Connector.open(String url, int mode);
• HttpsConnection httpsConn = (HttpsConnection)Connector.open(String url, int mode, boolean timeout);

It should be noted that the url for the HttpsConnection will be of the format https://www.site.com. The HttpsConnection obtained in step 1, can be used to read and write in the same way as the HttpConnection was in previous section. It should be noted that the same time out values mentioned in previous section for HttpConnection object (in step 1), read() (in step2) and write() (in step 3), will hold for HttpsConnection object, read()

There are now following services
- Web Application Review
- External Black Box Penetration Testing
- White Box Security Assessment
- Security Architecture Review
- Security Remediation Consulting

The very first question a Web server administrator must confront is, "Do I really want/need to provide dynamic content from my server?" While dynamic content has allowed for a diverse range of user interaction and become a de facto standard for most large Web sites, it remains one of the largest security threats on the Internet. CGI applications and SSI-enabled pages are not inherently insecure,

but poorly written code can potentially open up dangerous back doors and gaping holes on what would otherwise be a well-secured system.
The following are the three most common security risks CGI applications and SSI pages create:

- Information leaks: Providing any kind of system information to a hacker could potentially provide a hacker with the ammunition they need to break into your server. The less a hacker knows about the configuration of a system, the harder it is to break into.
- Access to potentially dangerous system commands/applications: One of the most common exploits used by hackers is to "take over" a service running on the server and use it for their own purposes. For example, gaining access to a mail application via an HTML form-based script, and then harnessing the mail server to send out spam or acquire confidential user information.
- Depleting system resources: While not a direct security threat per se, a poorly written CGI application can use up a system's available resources to the point where it becomes almost completely unresponsive.

1.3 HTTP and HTTPS difference

The HyperText Transfer Protocol is an application layer protocol, which means it focuses on how information is presented to the user of the computer but doesn't care a whit about how data gets from Point A to Point B. It is stateless, which means it doesn't attempt to remember anything about the previous Web session. This is great because there is less data to send, and that means speed. And HTTP operates on Transmission Control Protocol (TCP) Port 80 by default, meaning your computer must send and receive data through this port to use HTTP. Not just any old port will do.

Secure HyperText Transfer Protocol (HTTPS) is for all practical purposes HTTP. The chief distinction is that it uses TCP Port 443 by default, so HTTP and HTTPS are two separate communications. HTTPS works in conjunction with another protocol, Secure Sockets Layer (SSL), to transport data safely. Remember, HTTP and HTTPS don't care how the data gets to its destination. In contrast, SSL doesn't care what the data looks like. People often use the terms HTTPS and SSL interchangeably, but this isn't accurate. HTTPS is secure because it uses SSL to move data.

Hypertext Transfer Protocol (http) is a system for transmitting and receiving information across the Internet. Http serves as a request and response procedure that all agents on the Internet follow so that information can be rapidly, easily, and accurately disseminated between servers, which hold information, and clients, who are trying to access it. Http is commonly used to access html pages, but

other resources can be utilized as well through http. While exchanging confidential information with a server, which needs to be secured in order to prevent unauthorized access a client needs some sort of security which is provided by https, or secure http which allows authorization and secured transactions.

If you visit a website or webpage, and look at the address in the web browser, it will most likely begin with the following: http://. This means that the website is connected to your browser using the regular unsecure language, due to which there is a possiblilty for someone to spy on your computer's conversation with the website. If you fill out a form on the website, someone might see the information you send to that site.

But if the web address begins with https://, that basically means your computer is talking to the website in a secure code that no one can spy on the information you fill in.

https is quite similar to http, because it follows the same basic protocols. The http or https client, such as a Web browser, establishes a connection to a server on a standard port. When a server receives a request, it returns a status and a message, which may contain the requested information or indicate an error if part of the process malfunctioned. Both systems use the same Uniform Resource Identifier (URI) scheme, so that resources can be universally identified. Use of https in a URI scheme rather than http indicates that an encrypted connection is desired.

When using an https connection, the server responds to the initial connection by offering a list of encryption methods it supports. In response, the client selects a connection method, and the client and server exchange certificates to authenticate their identities. After this is done, both parties exchange the encrypted information after ensuring that both are using the same key, and the connection is closed. In order to host https connections, a server must have a public key certificate, which embeds key information with a verification of the key owner's identity. Most certificates are verified by a third party so that clients are assured that the key is secure.

Https is used in many situations, such as log-in pages for banking, forms, corporate log ons, and other applications in which data needs to be secure. However, if not implemented properly, https is not infallible, and therefore it is extremely important for end users to be wary about accepting questionable certificates and cautious with their personal information while using the Internet.

The importance of electronic commerce is widely acknowledged. Surveys of Web users indicate that poor performance is a major cause of dissatisfaction. This paper presents results that show that popular servers that use the Secure HyperText Transport Protocol (HTTPS), the secure version of the

HyperText Transport Protocol (HTTP), can transmit typical documents with little performance penalty.

Encrypted Communications

We briefly review the operation of secure Web communications. The Secure Sockets Layer (SSL) protocol has become most widely used method for encrypting and authenticating Web communications. Conducting SSL communications involves the following steps:

- A client establishes a Transmission Control Protocol (TCP) connection with a server, which involves one round-trip message delay when no failure occurs.
- On top of TCP, the client and server establish a secure SSL communication channel. The application implements this by issuing an SSL_connect() call to the SSL library. The client and server negotiate a mutually agreeable cipher, which is a stream encryption algorithm and an authentication method pair. The client and server use a public key cryptographic protocol to exchange secret session keys that will be used to encrypt and decrypt application layer messages. Secret keys are frequently used for streaming encryption because public key encryption is much more expensive.
- On top of SSL, the client and server exchange one or more HTTP messages. A single HTTP message is exchanged in HTTP/1.0; multiple messages would be exchanged in keep-alive or persistent connections. To use encryption the application code issues calls to SSL_write() and SSL_read(), instead of calls to TCP socket write() and read(), respectively.

The performance of the encryption algorithm RC4 [RC4] was studied. RC4 is a variable key-size stream cipher designed by Ron Rivest. Designed for byte-oriented operations, RC4 is expected to run quickly, as it uses only 8 to 16 instructions per byte.

A non-statistical survey of 5384 secure Web sites worldwide in March 1997 found that 59% of the secure servers were Netscape or Microsoft and used RC4 encryption. The portion for each server and key size is shown in Table 1.

	40 bit	128 bit	Total
Netscape-Enterprise (includes multiple versions)	16.4%	10.4%	26.8%

Microsoft-IIS (includes multiple versions)	20.0%	12.6%	32.6%
Total	36.4%	23.0%	59.4%

Table 1.1. The Subset of the Secure Servers Which Run Netscape or Microsoft and Use RC4

Secret key sizes of 40 and 128 bits constitute the vast majority of production RC4 keys. The smaller, 40-bit key is called "export strength" because the US Government permits its export from the United States. It is breakable with moderate effort. The 128-bit key is considered long enough to be unbreakable by known methods in typical large computer facilities -assuming it is used properly.

A small Intranet environment - A 10 Mbps Ethernet connected 2 PCs. Each PC was directly attached to an unswitched hub. The Web servers ran on a PC with a 2 GHz Pentium, 1 GB RAM and a fast Ethernet card, running NT. The clients ran on a PC, also running NT, with a 1 GHz Pentium with 512 MB RAM and an NE 2000 NIC Ethernet card.

Performance was measured in a no-load situation-during the experiments the PCs were otherwise unused and the hub was lightly used. In addition, during the experiments neither machine paged virtual memory.

We call our measurement apparatus WebPerf. WebPerf consists of a Web robot client and a back-end database. The robot measures Web response times and other parameters and stores the results in the database. The robot was written by one of us-Buff-in C++ and compiled with Visual C++ version with optimization. It communicates via Winsock.

To minimize contention with itself the robot browser runs single-threaded on an otherwise idle machine.

A widespread SSL implementation was integrated into the robot by one of us-Schmitt. SSL version 0.8.1 was used. It supports SSL versions 3.0. The robot does not authenticate the server since this is a client side activity.

Netscape Enterprise Server and Microsoft IIS were both installed on the server PC. Two identical Web sites were created with 24 documents of sizes 1,000, 2,000, 3,000, ..., 20,000, 40,000, 60,000, 80,000, 100,000 bytes. This size distribution is similar to that of documents requested on the Web as observed in traces from an Internet Service Provider and an Intranet in 1998. The documents contained typical HTML text (actually, the beginning of the HTTP/1.1 specification).

The robot browser sequentially requested each of the 24 documents many times. The robot measured the duration at the client between just before the client robot issued the SSL_write() of the request and just after the client completed the SSL_read() of the response. This measures the streaming application layer encryption performance. The costs of additional delays for establishing SSL connections were presented elsewhere.

This duration and many other measures are stored in an Oracle SQL database after the measurements have been made.

The word cryptography is derived from the two Greek words kryptós (“secret”) and gráfo (“write”) and is the science of message security. It is an ancient mathematical subject and has only recently become a branch of information security. By using a key it is possible to use a cipher to transform a readable message, plaintext, into an unreadable message, ciphertext. This procedure is known as encryption. The very same key or a related key can be used to do the inverse transformation, decryption. This part will introduce different cryptographic algorithms and describe some computer security fundamentals.

1.4 Cryptography

Cryptography is the task of transforming information into a form that is incomprehensible, but at the same time allows the intended recipient to retrieve the original information using a secret key. Cryptography is a Greek word that literally means the art of writing secrets. Cryptographic algorithms (or ciphers, as they are often called) are special programs designed to protect sensitive information on public communication networks. During encryption, ciphers transform the original plaintext message into unintelligible ciphertext. Decryption is the process of retrieving plaintext from ciphertext. Two forms of cryptography are commonly used in information systems today: secret-key ciphers and public-key ciphers. Secret-key ciphers (sometimes referred to as symmetric-key ciphers) use a single private key to encrypt and decrypt as illustrated in Figure 1. Public-key ciphers (or asymmetric-key ciphers) use a well-known public key to encrypt and require a different private key to decrypt. The process may also be reversed to produce what is known as a digital signature. Digital signatures authenticate the sender. Since only the person holding the private key knows its value, only that person can create a digital signature that others can decrypt with the public key.
The main purpose of the digital certificate is to ensure that the public key contained in the certificate belongs to the entity to which the certificate was issued.

Encryption techniques using public and private keys require a public-key infrastructure (PKI) to support the distribution and identification of public keys. Digital certificates package public keys, information about the algorithms used, owner or subject data, the digital signature of a Certificate Authority that has verified the subject data, and a date range during which the certificate can be considered valid.

Without certificates, it would be possible to create a new key pair and distribute the public key, claiming that it is the public key for almost anyone. You could send data encrypted with the private key and the public key would be used to decrypt the data, but there would be no assurance that the data was originated by anyone in particular. All the receiver would know is that a valid key pair was used.
Algorithms used for encryption/decryption fall into two categories: symmetric-key and public-key cryptography.

Symmetric-key Cryptography

In symmetric-key algorithms, the sender and receiver have the same set of keys and use the same algorithms to encrypt and decrypt messages. The keys used for encryption and decryption can differ but there is a transform to go from one key to the other. Most of the cryptographic research this century has been into symmetric key technology. A symmetric key cipher uses a secret sequence of characters or key to scramble any amount of plaintext into a unique ciphertext. A different key, used to scramble the same plaintext, will yield a completely different ciphertext. The plaintext is recovered by rerunning the same algorithm with the same key over the ciphertext. There are two types of symmetric cipher in common usage: The block cipher takes a fixed length of plaintext (called the block size) and generates the same amount of ciphertext. If the total length of the plaintext is not a multiple of the block size, then padding data may be used to make up the difference on the last block of plaintext. The stream cipher converts plaintext to ciphertext one bit at a time.

Public-key Cryptography

Public-key cryptography is also known as asymmetric cryptography. Different keys are used in each direction. The sender and receiver both have two keys. One is public and is known by all and one is private and is only known by the keeper. This form of cryptography was introduced in the 1970s and solves some of the problems with symmetric-keys. Public-key cryptosystems differ from their symmetric key counterparts in that they use a private key for decrypting and a public key for encrypting rather than having a single key to do both. Although the public and private keys are

mathematically related, the private key cannot feasibly be computed from the public key, so making it safe to publish the public key anywhere. This is achieved by relating the two keys through a mathematical .trap-door. function. A trap-door function is one which is easy to compute in one direction, but which has an inverse which is much more difficult to calculate. So in the case of public-key cryptography, creating the two keys from scratch is easy but attempting to find one from the other is hard. In the Diffie-Hellman system, the trapdoor function used is the discrete logarithm. Whereas the RSA setup uses factorisation of large numbers. It is important to realise that although both these problems are thought to be as hard as each other, no one has ever proved that they are mathematically .hard. to break. On the other hand no one has ever proved that they are mathematically .easy. to find either. A theoretical breakthrough along either path would significantly strengthen or weaken public-key cryptography. RSA relies on the use of large numbers for its security, these large numbers are measured in bit length. A similar discussion to that in the symmetric cipher realm exists concerning how many bits make up a secure RSA key [RIV93]. Recently a 129 digit number was factorised in an 8 month distributed software effort; 129 digits equates to a 429 bit key. It is now recommended people use at least 700 bit keys for security. Encryption and decryption are also inverse operations, no matter which order they are carried out in. This is useful for digital signatures

Digital Certificate

To verify that a certain public key belongs to an identity, a digital certificate can be used. A digital certificate binds a public key to an identity. To be able to trust that the sender of a certificate is the real owner, a third party needs to be involved. A trusted third party can sign a certificate by encrypting it with a private key. The third part is usually a Certificate Authority (CA). A CA is an organization that provides signing of certificates. It is also possible for the certificate owner to sign the certificate himself. A signed certificate contains an encrypted checksum, signature. A commonly used certificate format is the Directory Authentication Framework (X.509). X.509 defines what should be present in a certificate. This includes the identity, public key information of the issuer and owner and validity interval. It is validated by obtaining the public key of the issuer and decrypting the signature. A hash of all other data is calculated and should match the decrypted signature. The validity interval is checked to ensure that this certificate has not expired. The user can now use the public key of the certificate owner to continue the communication.

Server Certificates:

Server certificates identify servers that participate in secure communications with other computers using communication protocols such as SSL. These certificates allow a server to verify its identity to

clients. Server certificates follow the X.509 certificate format that is defined by the Public-Key Cryptography Standards (PKCS).

Software Publisher Certificates:
Microsoft Authenticode does not guarantee that signed code is safe to run, but rather informs the user whether or not the publisher is participating in the infrastructure of trusted publishers and CAs. These certificates are used to sign software to be distributed over the Internet.
Authenticode requires a software publisher certificate to sign Microsoft ActiveX and other compiled code. Internet Explorer is also capable of trusting software that is signed with a publisher's certificate.

To view a list of trusted software publishers in Internet Explorer, click Internet Options on the Tools menu, click the Content tab, and then click Publishers. You can also remove trusted publishers by clicking Remove in this screen.

Certificate Authority Certificates:
Internet Explorer 5 divides CAs into two categories, Root Certification Authorities and Intermediate Certification Authorities. Root certificates are self-signed, meaning that the subject of the certificate is also the signer of the certificate. Root Certification Authorities have the ability to assign certificates for Intermediate Certification Authorities. An Intermediate Certification Authority has the ability to issue server certificates, personal certificates, publisher certificates, or certificates for other Intermediate Certification Authorities.

For example, if you click Certificates on the Content tab in the Internet Explorer Properties dialog box, a list of certificates that are installed on your computer is displayed. There is a trusted Root Authority listed as "Class 1 Public Primary Certification Authority" (which is run by VeriSign). This certificate is issued and signed by the Class 1 Public Primary Certificate Authority, and is therefore a root certificate. On the Intermediate Certification Authorities tab, there are several certificates listed as "VeriSign Class 1 CA." The root certificate mentioned above issued these certificates. These Intermediate Certificate Authorities were created for the purpose of issuing and validating personal digital certificates, so if a person has obtained a Class 1 personal digital certificate from VeriSign, it will be issued by one of these Intermediate CAs. This creates what is known as a verification chain. In this case, there are only three certificates in the verification chain for a personal certificate. However, verification chains can contain a large number of certificates depending upon the number of Intermediate Certification Authorities in the chain.

1.5 Hash Functions

A hash function takes an input of arbitrary size and produces an output, a message digest or checksum, of fixed size. The most important property of a hash function is that the original message cannot be generated from the message digest. Another important property is that two different messages have a very low probability of resulting in the same message digest, a collision. Hash functions can be used to add a message digest to the end of a sent message. The receiver can, if the algorithm for creating the message digest is known, verify the message digest. If the receiver calculates the message digest from the message and gets the same result, the message can be assumed to be free from accidental, but not necessarily of intentional modifications. A cryptographic hash function is a deterministic procedure that takes an arbitrary block of data and returns a fixed-size bit string, the (cryptographic) hash value, such that an accidental or intentional change to the data will change the hash value. The data to be encoded is often called the "message", and the hash value is sometimes called the message digest or simply digest.

The ideal cryptographic hash function has four main properties:
it is easy to compute the hash value for any given message,
it is infeasible to find a message that has a given hash,
it is infeasible to modify a message without changing its hash,
it is infeasible to find two different messages with the same hash.

The basic requirements for a cryptographic hash function are:
the input can be of any length,
the output has a fixed length,
H(x) is relatively easy to compute for any given x ,
H(x) is one-way,
H(x) is collision-free.

Cryptographic hash functions have many information security applications, notably in digital signatures, message authentication codes (MACs), and other forms of authentication. They can also be used as ordinary hash functions, to index data in hash tables; as fingerprints, to detect duplicate data or uniquely identify files; or as checksums to detect accidental data corruption. Indeed, in information security contexts, cryptographic hash values are sometimes called (digital) fingerprints, checksums, or

just hash values, even though all these terms stand for functions with rather different properties and purposes.

Properties

Most cryptographic hash functions are designed to take a string of any length as input and produce a fixed-length hash value.

A cryptographic hash function must be able to withstand all known types of cryptanalytic attack. As a minimum, it must have the following properties:
Preimage resistance: given a hash h it should be hard to find any message m such that h = hash(m). This concept is related to that of one way function. Functions that lack this property are vulnerable to preimage attacks.
Second preimage resistance: given an input m1, it should be hard to find another input, m2 (not equal to m1) such that hash(m1) = hash(m2). This property is sometimes referred to as weak collision resistance. Functions that lack this property are vulnerable to second preimage attacks.
Collision resistance: it should be hard to find two different messages m1 and m2 such that hash(m1) = hash(m2). Such a pair is called a (cryptographic) hash collision, and this property is sometimes referred to as strong collision resistance. It requires a hash value at least twice as long as what is required for preimage-resistance, otherwise collisions may be found by a birthday attack.

These properties imply that a malicious adversary can not replace or modify the input data without changing its digest. Thus, if two strings have the same digest, one can be very confident that they are identical.

A function meeting these criteria may still have undesirable properties. Currently popular cryptographic hash functions are vulnerable to length-extension attacks: given h(m) and len(m) but not m, by choosing a suitable m' an attacker can calculate h (m || m'), where || denotes concatenation. This property can be used to break naive authentication schemes based on hash functions. The HMAC construction works around these problems.

Ideally, one may wish for even stronger conditions. It should be impossible for an adversary to find two messages with substantially similar digests; or to infer any useful information about the data, given only its digest. Therefore, a cryptographic hash function should behave as much as possible like a random function while still being deterministic and efficiently computable.

Checksum algorithms, such as CRC32 and other cyclic redundancy checks, are designed to meet much weaker requirements, and are generally unsuitable as cryptographic hash functions. For example, a CRC was used for message integrity in the WEP encryption standard, but an attack was readily discovered which exploited the linearity of the checksum.

Applications

A typical use of a cryptographic hash would be as follows: Alice poses a tough math problem to Bob, and claims she has solved it. Bob would like to try it himself, but would yet like to be sure that Alice is not bluffing. Therefore, Alice writes down her solution, appends a random nonce, computes its hash and tells Bob the hash value (whilst keeping the solution and nonce secret). This way, when Bob comes up with the solution himself a few days later, Alice can prove that she had the solution earlier by revealing the nonce to Bob. (This is an example of a simple commitment scheme; in actual practice, Alice and Bob will often be computer programs, and the secret would be something less easily spoofed than a claimed puzzle solution).

Another important application of secure hashes is verification of message integrity. Determining whether any changes have been made to a message (or a file), for example, can be accomplished by comparing message digests calculated before, and after, transmission (or any other event).

A message digest can also serve as a means of reliably identifying a file; several source code management systems, including Git, Mercurial and Monotone, use the sha1sum of various types of content (file content, directory trees, ancestry information, etc) to uniquely identify them.

A related application is password verification. Passwords are usually not stored in cleartext, for obvious reasons, but instead in digest form. To authenticate a user, the password presented by the user is hashed and compared with the stored hash. This is sometimes referred to as one-way encryption.

For both security and performance reasons, most digital signature algorithms specify that only the digest of the message be "signed", not the entire message. Hash functions can also be used in the generation of pseudorandom bits.

Hashes are used to identify files on peer-to-peer filesharing networks. For example, in an ed2k link, an MD4-variant hash is combined with the file size, providing sufficient information for locating file

sources, downloading the file and verifying its contents. Magnet links are another example. Such file hashes are often the top hash of a hash list or a hash tree which allows for additional benefits.

[edit]
Hash functions based on block ciphers

There are several methods to use a block cipher to build a cryptographic hash function, specifically a one-way compression function.

The methods resemble the block cipher modes of operation usually used for encryption. All well-known hash functions, including MD4, MD5, SHA-1 and SHA-2 are built from block-cipher-like components designed for the purpose, with feedback to ensure that the resulting function is not bijective.

A standard block cipher such as AES can be used in place of these custom block ciphers; this generally carries a cost in performance, but can be advantageous where a system needs to perform hashing and another cryptographic function such as encryption that might use a block cipher, but is constrained in the code size or hardware area it must fit into, such as in some embedded systems like smart cards.

1.6 Cryptographic hash algorithms

There is a long list of cryptographic hash functions, although many have been found to be vulnerable and should not be used. Even if a hash function has never been broken, a successful attack against a weakened variant thereof may undermine the experts' confidence and lead to its abandonment. For instance, in August 2004 weaknesses were found in a number of hash functions that were popular at the time, including SHA-0, RIPEMD, and MD5. This has called into question the long-term security of later algorithms which are derived from these hash functions — in particular, SHA-1 (a strengthened version of SHA-0), RIPEMD-128, and RIPEMD-160 (both strengthened versions of RIPEMD). Neither SHA-0 nor RIPEMD are widely used since they were replaced by their strengthened versions.

As of 2009, the two most commonly used cryptographic hash functions are MD5 and SHA-1. However, MD5 has been broken; an attack against it was used to break SSL in 2008.

SHA-0 and SHA-1 are members of the SHA family of hash functions developed by the NSA. In February 2005, a successful attack on SHA-1 was reported, finding collisions in about 269 hashing

operations, rather than the 280 expected for a 160-bit hash function. In August 2005, another successful attack on SHA-1 was reported, finding collisions in 263 operations. Theoretical weaknesses of SHA-1 exist as well, suggesting that it may be practical to break within years. New applications can avoid these problems by using more advanced members of the SHA family, such as SHA-2, or using techniques such as randomized hashing that do not require collision resistance.

However, to ensure the long-term robustness of applications that use hash functions, there is a competition to design a replacement for SHA-2, which will be given the name SHA-3 and become a FIPS standard around 2012.

Some of the following algorithms are known to be insecure; consult the article for each specific algorithm for more information on the status of each algorithm. Note that this list does not include candidates in the current NIST hash function competition. For additional hash functions see the box at the bottom of the page.

Chapter 2.

Background

2.1. HTTP and HTTPS connection

We can create HTTP connections in Java applications using the HTTP protocol handling code built in to the Java Developer's Kit, and HTTPS connections using the HTTPS protocol handler provided with EAServer.

HTTP connections

The standard Java virtual machine provides HTTP connectivity with these classes in java.net package:

URL allows you to use Uniform Resource Locator strings for HTTP connections and other protocol connections that can be represented by URLs.

- URLConnection represents a connection to a server and resource indicated by a URL.
- HttpURLConnection extends URL with additional methods that are specific to the HTTP protocol.

The following code shows a typical example. This code opens a connection, retrieves the data (text is assumed), and prints it:

```
URL url = new URL("http://www.site.com/");
URLConnection conn = url.openConnection();
conn.connect();
InputStreamReader content
   = new InputStreamReader(conn.getInputStream());
```

```
for (int i=0; i != -1; i = content.read())
{
   System.out.print((char) i);
}
```

HTTPS connections

The procedure for creating HTTPS connections is similar to that for HTTP connections, except that you must install EAServer's HTTPS protocol handler in the Java virtual machine and configure SSL parameters before opening a connection.

System requirements EAServer's HTTPS protocol handler uses the same SSL implementation as used by Java and C++ clients and requires a full client runtime install.

Installing the HTTPS protocol handler

The EAServer HTTPS protocol handler can be installed two ways:

- By configuring the java.protocol.handler.pkgs Java system property, making it the default handler for all HTTPS URLs. This is the recommended approach if you do not need to use another vendor's HTTPS protocol handler in addition to the EAServer implementation.
- By calling one of the java.net.URL constructors that takes a java.net.URLStreamHandler as a parameter. This approach must be used if you must use more than one HTTPS protocol handler in one EAServer or in one client application.

Configuring the default protocol handlers

The java.protocol.handler.pkgs Java system property configures the Java virtual machine default URL protocol handlers. To use the EAServer handlers, you must add com.sybase.jaguar.net to the list.

In a client application, specify this property on the command line; for example:

```
jre -Djava.protocol.handler.pkgs=com.sybase.jaguar.net ...
```

For an EAServer, set the JVM options property using the Advanced tab in the Server Properties dialog box:

Property	Value
com.sybase.jaguar.server.jvm.options	If not already set, set to: -java.protocol.handler.pkgs=com.sybase.jaguar.net If already set, verify that the value includes this option. JVM options must be separated with a comma.

You can specify more than one package by separating package names with a character, but you can configure only one handler per protocol. If you must use more than one HTTPS protocol handler in one EAServer or in one client application, you must call one of the java.net.URL constructors that takes a java.net.URLStreamHandler as a parameter. The specified java.net.URLStreamHandler instance overrides the default handler for the protocol specified by the URL. For example, to specify the EAServer HTTPS handler, use code like this:

```
import java.net.*;
import com.sybase.jaguar.net.JagURLStreamHandlerFactory;
import com.sybase.jaguar.net.HttpsURLConnection;
....
String url_string = "https://localhost:8081/index.html";
// The URL stream handler factory is required to create a stream
// handler.
JagURLStreamHandlerFactory fact = new JagURLStreamHandlerFactory();
// Extract the protocol from the front of the URL string
String protocol = url_string.substring(0, url_string.indexOf(":"));
```

```
// If the protocol is HTTPS, use the EAServer HTTPS handler. Otherwise,
// use the default handler
java.net.URL url;
if (protocol.equals("https"))
{
   url = new URL((URL)null, url_string,
      fact.createURLStreamHandler(protocol));
} else
{
   url = new URL(url_string);
}
```

EAServer's HttpsURLConnection class

EAServer provides the com.sybase.jaguar.net.HttpsURLConnection class to support HTTPS connectivity. This class extends java.net.URLConnection and implements all methods of java.net.HttpURLConnection. HttpsURLConnection provides these additional methods specifically for SSL support:

- A setSSLProperty method with signature:
- void setSSLProperty (String prop, String value) throws
- CtsSecurity.InvalidPropertyException,
- CtsSecurity.InvalidValueException

Call this method to set the SSL properties described in "SSL properties".

- A setSSLProperties method with signature:
- void setSSLProperty (java.util.Properties props) throws
- CtsSecurity.InvalidPropertyException,
- CtsSecurity.InvalidValueException

This method is the same as setSSLProperty, but allows you to set multiple properties with one call.

- A getSSLProperty method with signature:
- String[] setSSLProperty (String prop) throws
- CtsSecurity.InvalidPropertyException

Call this method to retrieve the SSL properties described in “SSL properties”.

- A setGlobalProperty method with signature:
- void setGlobalProperty (String prop, String value) throws
- CtsSecurity.InvalidPropertyException,
- CtsSecurity.InvalidValueException

Properties set with this method affect the handling of all HTTPS connections, not just the current one.

- A getGlobalProperty method with signature:
- String[] getGlobalProperty(String prop) throws
- CtsSecurity.InvalidPropertyException;

Call this method to retrieve the global SSL properties described in “SSL properties”.

- A getSessionInfo method with signature:
- CtsSecurity.SSLSessionInfo getSessionInfo() throws CtsSecurity.SSLException

The SSLSessionInfo methods allow you to determine the SSL session properties, such as the server’s address, the client certificate in use, the server certificate in use, and so forth. For more information, see the Interface Repository documentation for the CtsSecurity::SSLSessionInfo IDL interface. getSessionInfo throws an a SSLException instance if SSL is not used on the connection.

Creating HTTPS connections

1. Configure or install the EAServer HTTPS protocol handler as described in “Installing the HTTPS protocol handler”.
2. Create URL and URLConnection instances. If connecting to an EAServer, specify the address of an HTTPS listener that supports the desired level of security. For example:
3. URL url = new URL("https://myhost:8081/index.html");
4. URLConnection conn = url.openConnection();

5. Verify that the object returned by URL.openConnection is of class com.sybase.jaguar.net.HttpsURLConnection, then set SSL properties for the connection. “SSL properties” describes the SSL properties that can be set. At a minimum, you must specify the qop and pin properties, as well as the certificateLabel property if using mutual authentication. For example:

```
6.  if (conn instanceof HttpsURLConnection)
7.  {
8.   HttpsURLConnection https_conn =  (HttpsURLConnection) conn;
9.   try
10.  {
11.   https_conn.setSSLProperty( "qop","sybpks_intl" );
12.   https_conn.setSSLProperty( "pin", "secret");
13.   https_conn.setSSLProperty(
14.        "certificateLabel", "John Smith");
15.  }
16.  catch ( CtsSecurity.InvalidPropertyException ipe )
17.  {
18.   System.err.println( ipe );
19.  }
20.  catch ( CtsSecurity.InvalidValueException ive )
21.  {
22.   System.err.println( ive );
23.  }
```

24. Open the connection, for example:

```
25.  conn.connect();
```

Once the connection is open, you can perform any valid operation for a connection that uses java.net.HTTPUrlConnection. You can also call the getSessionInfo method to retrieve a CtsSecurity.SSLSessionInfo instance that allows you to verify the SSL connection parameters. For example:

```
java.net.URLConnection conn;
```

```
... deleted code that constructed URLConnection ...

if (conn instanceof HttpsURLConnection)

{

  HttpsURLConnection https_conn =     (HttpsURLConnection) conn;

  CtsSecurity.SSLSessionInfo sessInfo =

    https_conn.getSessionInfo();
```

The SSLSessionInfo methods allow you to determine the SSL session properties, such as the server's address, the client certificate in use, the server certificate in use, and so forth. For more information, see the Interface Repository documentation for the CtsSecurity::SSLSessionInfo interface.

2.2 SSL

Global properties are set and read with the getGlobalProperty and setGlobalProperty methods. Global properties affect all HTTPS connections, not just the HttpsUrlConnection instance on which they are set. The right column in Table 3 lists which methods are valid for each property.

Some properties, if not set or set incorrectly, cause the connection to invoke an SSL callback method. You can install a callback to respond to these cases with the callbackImpl global property. If you do not install an SSL callback, the default callback implementation aborts the connection attempt. To use SSL, you must specify the name of an available security characteristic as the value for the qop property. The characteristic describes the CipherSuites the client uses when negotiating an SSL connection. When connecting, the client sends the list of CipherSuites that it uses to the server, and the server selects a cipher suite from that list. The server chooses the first cipher suite in the list that it can use. If the server cannot use any of the available CipherSuites, the connection fails.

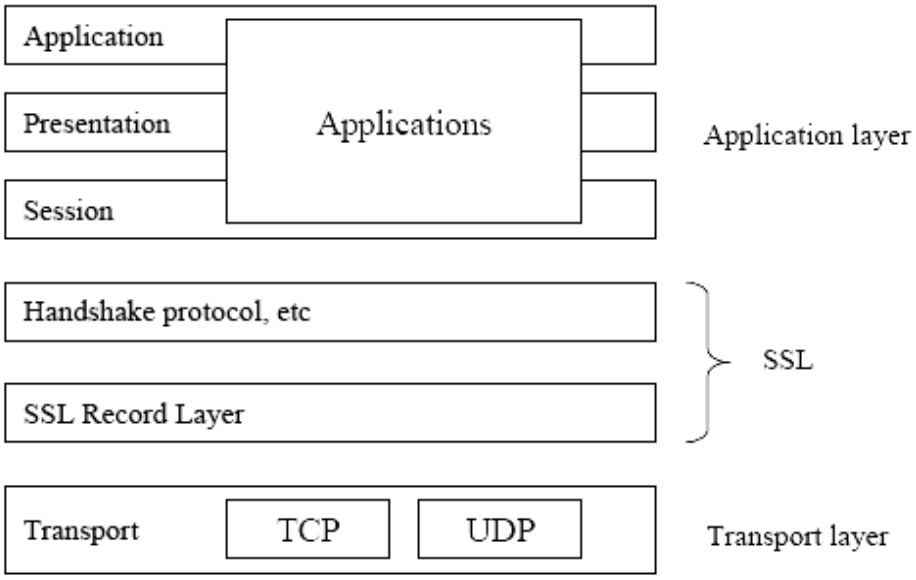


Figure 2.1 Insertion of SSL protocols

The Transmission Control Protocol/Internet Protocol (TCP/IP) governs the transport and routing of data over the Internet. Other protocols, such as the HyperText Transport Protocol (HTTP), Lightweight Directory Access Protocol (LDAP), or Internet Messaging Access Protocol (IMAP), run "on top of" TCP/IP in the sense that they all use TCP/IP to support typical application tasks such as displaying web pages or running email servers.

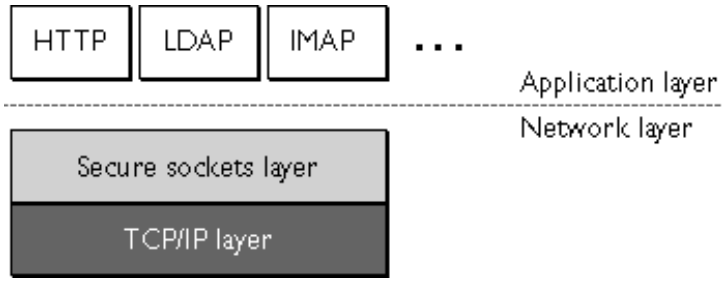


Figure 2.2 SSL runs above TCP/IP and below high-level application protocols

The SSL protocol runs above TCP/IP and below higher-level protocols such as HTTP or IMAP. It uses TCP/IP on behalf of the higher-level protocols, and in the process allows an SSL-enabled server to authenticate itself to an SSL-enabled client, allows the client to authenticate itself to the server, and allows both machines to establish an encrypted connection.

These capabilities address fundamental concerns about communication over the Internet and other TCP/IP networks:

- SSL server authentication allows a user to confirm a server's identity. SSL-enabled client software can use standard techniques of public-key cryptography to check that a server's certificate and public ID are valid and have been issued by a certificate authority (CA) listed in the client's list of trusted CAs. This confirmation might be important if the user, for example, is sending a credit card number over the network and wants to check the receiving server's identity.
- SSL client authentication allows a server to confirm a user's identity. Using the same techniques as those used for server authentication, SSL-enabled server software can check that a client's certificate and public ID are valid and have been issued by a certificate authority (CA) listed in the server's list of trusted CAs. This confirmation might be important if the server, for example, is a bank sending confidential financial information to a customer and wants to check the recipient's identity.
- An encrypted SSL connection requires all information sent between a client and a server to be encrypted by the sending software and decrypted by the receiving software, thus providing a high degree of confidentiality. Confidentiality is important for both parties to any private transaction. In addition, all data sent over an encrypted SSL connection is protected with a mechanism for detecting tampering--that is, for automatically determining whether the data has been altered in transit.

The SSL protocol includes two sub-protocols: the SSL record protocol and the SSL handshake protocol. The SSL record protocol defines the format used to transmit data. The SSL handshake protocol involves using the SSL record protocol to exchange a series of messages between an SSL-enabled server and an SSL-enabled client when they first establish an SSL connection. This exchange of messages is designed to facilitate the following actions:

- Authenticate the server to the client.
- Allow the client and server to select the cryptographic algorithms, or ciphers, that they both support.
- Optionally authenticate the client to the server.
- Use public-key encryption techniques to generate shared secrets.
- Establish an encrypted SSL connection.

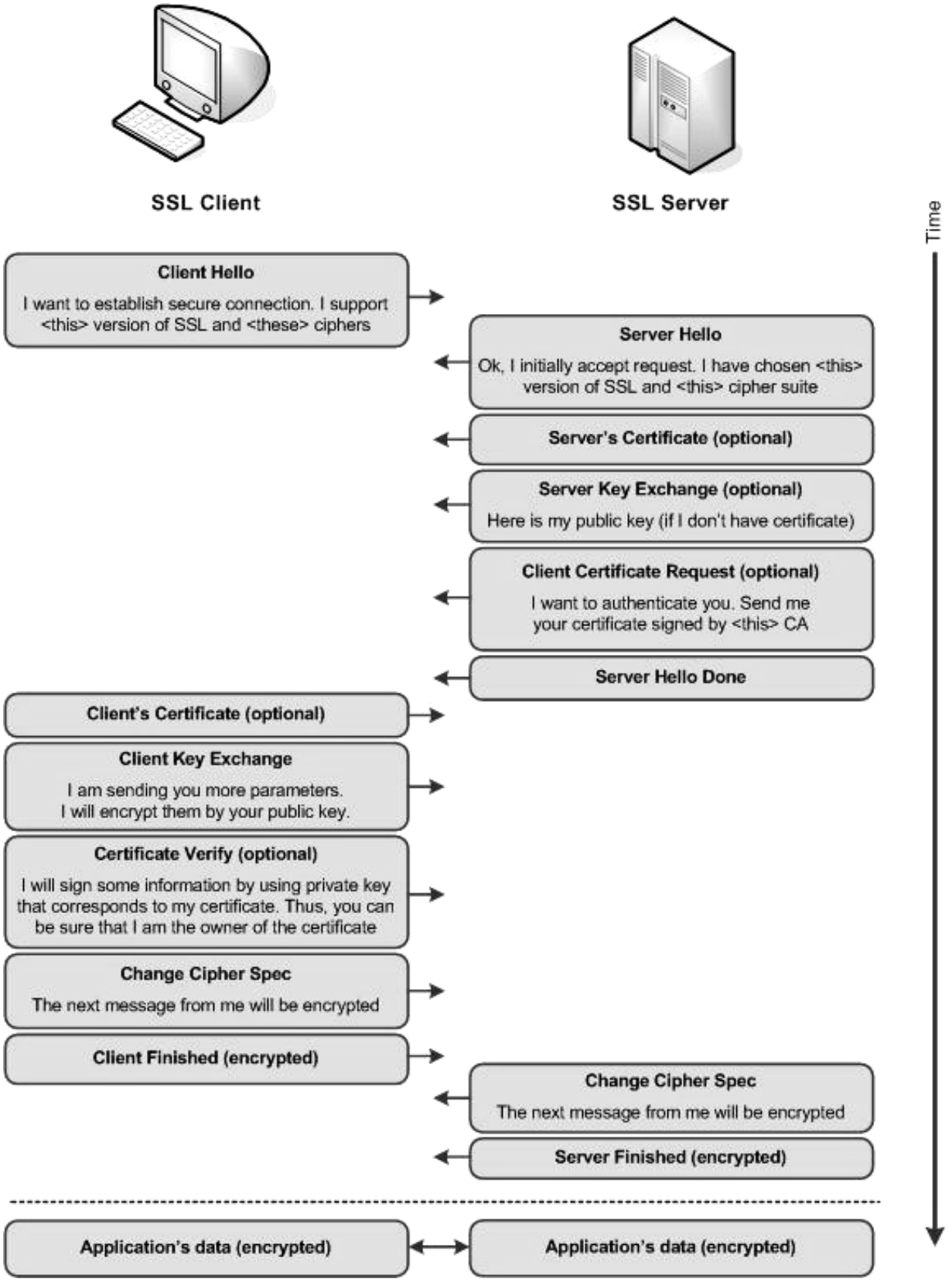


Figure 2.3 SSL protocols installations

2.3 The SSL Handshake

The SSL protocol uses a combination of public-key and symmetric key encryption. Symmetric key encryption is much faster than public-key encryption, but public-key encryption provides better authentication techniques. An SSL session always begins with an exchange of messages called the SSL handshake. The handshake allows the server to authenticate itself to the client using public-key techniques, then allows the client and the server to cooperate in the creation of symmetric keys used for rapid encryption, decryption, and tamper detection during the session that follows. Optionally, the handshake also allows the client to authenticate itself to the server.

The exact programmatic details of the messages exchanged during the SSL handshake are beyond the scope of this document. However, the steps involved can be summarized as follows (assuming the use of the cipher suites listed in Cipher Suites with RSA Key Exchange):

1. The client sends the server the client's SSL version number, cipher settings, randomly generated data, and other information the server needs to communicate with the client using SSL.
2. The server sends the client the server's SSL version number, cipher settings, randomly generated data, and other information the client needs to communicate with the server over SSL. The server also sends its own certificate and, if the client is requesting a server resource that requires client authentication, requests the client's certificate.
3. The client uses some of the information sent by the server to authenticate the server (see Server Authentication for details). If the server cannot be authenticated, the user is warned of the problem and informed that an encrypted and authenticated connection cannot be established. If the server can be successfully authenticated, the client goes on to Step 4.
4. Using all data generated in the handshake so far, the client (with the cooperation of the server, depending on the cipher being used) creates the premaster secret for the session, encrypts it with the server's public key (obtained from the server's certificate, sent in Step 2), and sends the encrypted premaster secret to the server.
5. If the server has requested client authentication (an optional step in the handshake), the client also signs another piece of data that is unique to this handshake and known by both the client and server. In this case the client sends both the signed data and the client's own certificate to the server along with the encrypted premaster secret.
6. If the server has requested client authentication, the server attempts to authenticate the client (see Client Authentication for details). If the client cannot be authenticated, the session is terminated. If the client can be successfully authenticated, the server uses its private key to

decrypt the premaster secret, then performs a series of steps (which the client also performs, starting from the same premaster secret) to generate the master secret.

7. Both the client and the server use the master secret to generate the session keys, which are symmetric keys used to encrypt and decrypt information exchanged during the SSL session and to verify its integrity--that is, to detect any changes in the data between the time it was sent and the time it is received over the SSL connection.
8. The client sends a message to the server informing it that future messages from the client will be encrypted with the session key. It then sends a separate (encrypted) message indicating that the client portion of the handshake is finished.
9. The server sends a message to the client informing it that future messages from the server will be encrypted with the session key. It then sends a separate (encrypted) message indicating that the server portion of the handshake is finished.
10. The SSL handshake is now complete, and the SSL session has begun. The client and the server use the session keys to encrypt and decrypt the data they send to each other and to validate its integrity.

Before continuing with the session, Netscape servers can be configured to check that the client's certificate is present in the user's entry in an LDAP directory. This configuration option provides one way of ensuring that the client's certificate has not been revoked.

It's important to note that both client and server authentication involves encrypting some piece of data with one key of a public-private key pair and decrypting it with the other key:

- In the case of server authentication, the client encrypts the premaster secret with the server's public key. Only the corresponding private key can correctly decrypt the secret, so the client has some assurance that the identity associated with the public key is in fact the server with which the client is connected. Otherwise, the server cannot decrypt the premaster secret and cannot generate the symmetric keys required for the session, and the session will be terminated.
- In the case of client authentication, the client encrypts some random data with the client's private key--that is, it creates a digital signature. The public key in the client's certificate can correctly validate the digital signature only if the corresponding private key was used. Otherwise, the server cannot validate the digital signature and the session is terminated.

Message Type	**Direction**	**Data Transfered**
client-hello	C > S	challenge-data, cipher-specs
server-hello	C < S	connection-id, server-certificate, cipher-specs

client-master-key	C > S	cipher-kind, clear-master-key, {secret-master-key}server-public-key
client-finish	C > S	{connection-id}client-write-key
server-verify	C < S	{challenge-data}server-write-key
server-finish	C < S	{session-id}server-write-key

Table of SSL messages: no session-id, no client authentication

The client-hello message sends the server some challenge-data and a list of ciphers which the client can support. The challenge-data is used to authenticate the server later on. The server-hello message returns a connection-id, a server certificate and a modified list of ciphers which the client and server can both support. The server certificate is used by the client to obtain the servers public key and verify the identity of the server using any certification authority certificates it has.

The client-master-key message is sent in the clear with respect to the SSL record protocol. In this message, the client sends the final choice of cipher (chosen from the cipher list it received from the server-hello message) and the master key sent in two parts. If the chosen cipher is an export grade cipher then 88 bits of the master key are sent unencrypted and only 40 bits are sent encrypted with the servers public key. If on the other hand the client has selected a full strength cipher then the entire key is sent encrypted with the server's public key and the clear-master-key field is empty.

From this point in the handshaking all messages are encrypted at the SSL Record Protocol level. It should be noted that the master key is not actually used as a key for the bulk encryption directly, rather it is used to calculate two sets of keys which are. From now on when the client sends a message to the server, the client uses the client-write-key to encrypt and the server uses the server-read-key to decrypt (client-write-key is the same as server-readkey). Similarly when the server sends a message to the client, the server uses the server-writekey to encrypt and the client uses the client-read-key to decrypt (server-write-key is the same as client-read-key). It is these two sets of keys which are calculated from the client-masterkey. Some ciphers require more than one actual key to encrypt and decrypt , so the various write keys and read keys may consist of more than one actual key. This is the reason that the actual session key is not sent and a master key is used.

The client-finish message consists of the connection-id originally sent by the server. The connection-id acts as a nonce value which prevents certain attacks. The message is encrypted using the client-write-key. The server-verify message contains the challenge-data originally sent by the client. The receipt and decryption of this message is the final stage of server authentication in that only the real server should have the private key necessary to decrypt the secret-master-key in the client-master-key message. This message is encrypted using the server-write-key. Finally the server-finish message terminates the handshaking section. It contains a new piece of data generated by the server called the session-id. The session-id is used in subsequent handshakes between the same client and server to avoid having to go through all the cipher and master key negotiation again. Session-ids are cached by each party after a connection is closed and only reused for subsequent connections if they do not occur too far apart. It is recommended that a session-id have a 100 second life in a cache before being discarded.

2.4. Server Authentication

Netscape's SSL-enabled client software always requires server authentication, or cryptographic validation by a client of the server's identity. As explained in Step 2 of The SSL Handshake, the server sends the client a certificate to authenticate itself. The client uses the certificate in Step 3 to authenticate the identity the certificate claims to represent.

To authenticate the binding between a public key and the server identified by the certificate that contains the public key, an SSL-enabled client must receive a "yes" answer to the four questions shown in Figure 2. Although the fourth question is not technically part of the SSL protocol, it is the client's responsibility to support this requirement, which provides some assurance of the server's identity and thus helps protect against a form of security attack known as "man in the middle."

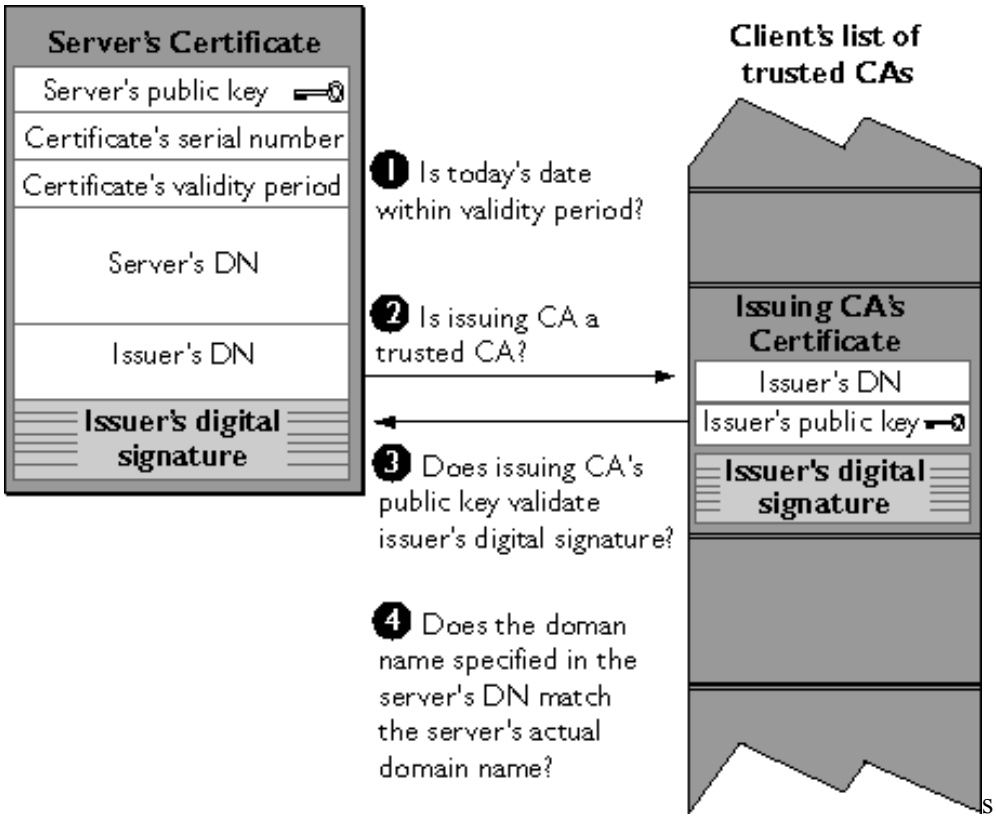


Figure 2.4 How a Netscape client authenticates a server certificate

An SSL-enabled client goes through these steps to authenticate a server's identity:

1. The client checks the server certificate's validity period. If the current date and time are outside of that range, the authentication process won't go any further. If the current date and time are within the certificate's validity period, the client goes on to Step 2.
2. Each SSL-enabled client maintains a list of trusted CA certificates, represented by the shaded area on the right side of Figure 2. This list determines which server certificates the client will accept. If the distinguished name (DN) of the issuing CA matches the DN of a CA on the client's list of trusted CAs, the answer to this question is yes, and the client goes on to Step 3. If the issuing CA is not on the list, the server will not be authenticated unless the client can verify a certificate chain ending in a CA that is on the list (see CA Hierarchies for details).
3. The client uses the public key from the CA's certificate (which it found in its list of trusted CAs in step 2) to validate the CA's digital signature on the server certificate being presented. If the information in the server certificate has changed since it was signed by the CA or if the CA certificate's public key doesn't correspond to the private key used by the CA to sign the server certificate, the client won't authenticate the server's identity. If the CA's digital signature can be validated, the server treats the user's certificate as a valid "letter of introduction" from that CA

and proceeds. At this point, the client has determined that the server certificate is valid. It is the client's responsibility to take Step 4 before Step 5.

4. This step confirms that the server is actually located at the same network address specified by the domain name in the server certificate. Although step 4 is not technically part of the SSL protocol, it provides the only protection against a form of security attack known as a Man-in-the-Middle Attack. Clients must perform this step and must refuse to authenticate the server or establish a connection if the domain names don't match. If the server's actual domain name matches the domain name in the server certificate, the client goes on to Step 5.
5. The server is authenticated. The client proceeds with the SSL handshake. If the client doesn't get to step 5 for any reason, the server identified by the certificate cannot be authenticated, and the user will be warned of the problem and informed that an encrypted and authenticated connection cannot be established. If the server requires client authentication, the server performs the steps described in Client Authentication.

After the steps described here, the server must successfully use its private key to decrypt the premaster secret the client sends in Step 4 of The SSL Handshake. Otherwise, the SSL session will be terminated. This provides additional assurance that the identity associated with the public key in the server's certificate is in fact the server with which the client is connected.

An SSL-enabled server goes through these steps to authenticate a user's identity:

1. The server checks that the user's digital signature can be validated with the public key in the certificate. If so, the server has established that the public key asserted to belong to John Doe matches the private key used to create the signature and that the data has not been tampered with since it was signed. At this point, however, the binding between the public key and the DN specified in the certificate has not yet been established. The certificate might have been created by someone attempting to impersonate the user. To validate the binding between the public key and the DN, the server must also complete Step 3 and Step 4.
2. The server checks the certificate's validity period. If the current date and time are outside of that range, the authentication process won't go any further. If the current date and time are within the certificate's validity period, the server goes on to Step 3.
3. Each SSL-enabled server maintains a list of trusted CA certificates, represented by the shaded area on the right side of Figure 3. This list determines which certificates the server will accept. If the DN of the issuing CA matches the DN of a CA on the server's list of trusted CAs, the answer to this question is yes, and the server goes on to Step 4. If the issuing CA is not on the

list, the client will not be authenticated unless the server can verify a certificate chain ending in a CA that is on the list (see CA Hierarchies for details). Administrators can control which certificates are trusted or not trusted within their organizations by controlling the lists of CA certificates maintained by clients and servers.

4. The server uses the public key from the CA's certificate (which it found in its list of trusted CAs in Step 3) to validate the CA's digital signature on the certificate being presented. If the information in the certificate has changed since it was signed by the CA or if the public key in the CA certificate doesn't correspond to the private key used by the CA to sign the certificate, the server won't authenticate the user's identity. If the CA's digital signature can be validated, the server treats the user's certificate as a valid "letter of introduction" from that CA and proceeds. At this point, the SSL protocol allows the server to consider the client authenticated and proceed with the connection as described in Step 6. Netscape servers may optionally be configured to take Step 5 before Step 6.
5. This optional step provides one way for a system administrator to revoke a user's certificate even if it passes the tests in all the other steps. The Netscape Certificate Server can automatically remove a revoked certificate from the user's entry in the LDAP directory. All servers that are set up to perform this step will then refuse to authenticate that certificate or establish a connection. If the user's certificate in the directory is identical to the user's certificate presented in the SSL handshake, the server goes on to step 6.
6. The server checks what resources the client is permitted to access according to the server's access control lists (ACLs) and establishes a connection with appropriate access. If the server doesn't get to step 6 for any reason, the user identified by the certificate cannot be authenticated, and the user is not allowed to access any server resources that require authentication. .

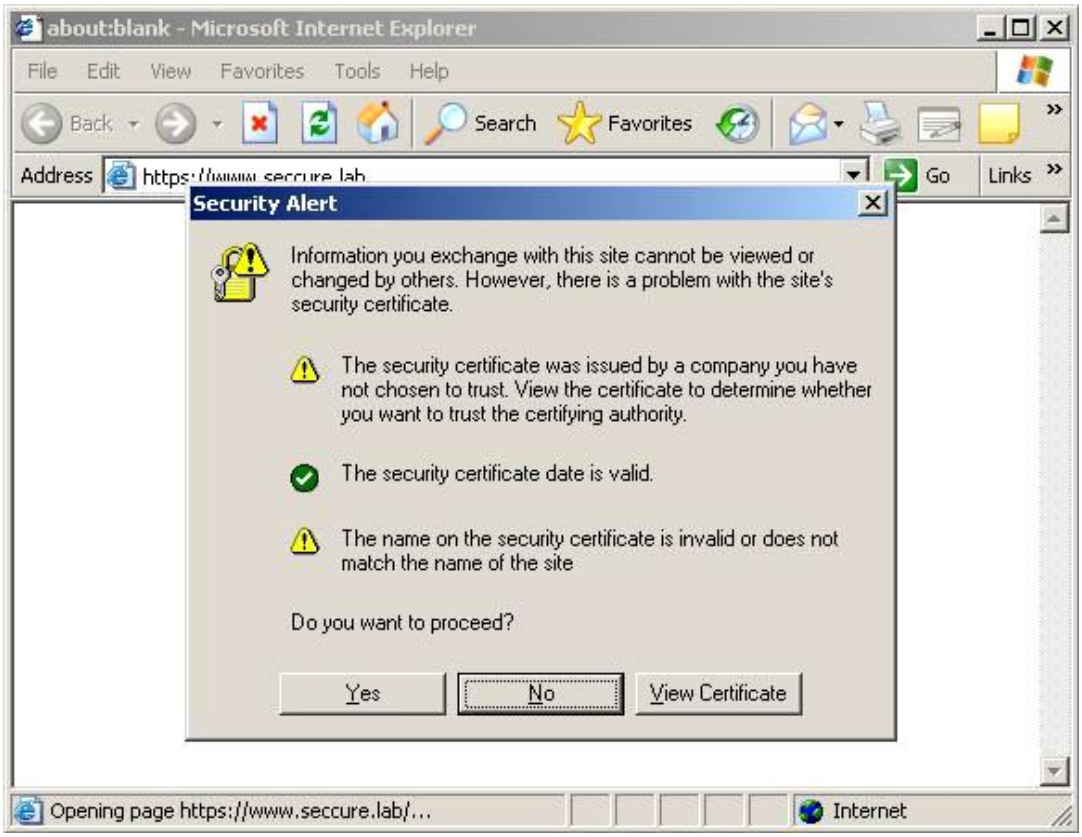


Figure 2.5 Exemplary reports about certificate.

Certificate of web server

Certificate:

Data:

Version: 3 (0x2)

Serial Number: 1 (0x1)

Signature Algorithm: sha1WithRSAEncryption

Issuer: O=Seccure, OU=Seccure Root CA

Validity

Not Before: Nov 28 01:00:20 2004 GMT

Not After : Nov 28 01:00:20 2005 GMT

Subject: O=Seccure, OU=Seccure Labs, CN=www.seccure.lab

Subject Public Key Info:

Public Key Algorithm: rsaEncryption

RSA Public Key: (1024 bit)

Modulus (1024 bit):

00:c1:19:c7:38:f4:89:91:27:a2:1b:1d:b6:8d:91:

```
            48:63:0e:3d:0d:2e:f8:65:45:56:db:98:4d:11:21:
            01:ac:81:8e:3f:64:4a:8a:3f:21:15:ca:49:6e:64:
            5c:5d:a2:ab:5a:48:cb:2a:9f:0c:02:b9:ff:52:f6:
            d9:39:6d:a3:4a:94:41:f9:e9:ab:f0:42:fb:68:9a:
            4b:53:41:e7:4f:b0:2b:02:d7:92:a2:2b:02:a2:f9:
            f1:2d:68:fa:50:01:2f:49:c1:28:2f:a8:c6:6d:6d:
            ab:1d:b9:bd:c9:80:63:f1:d6:22:19:de:2d:4a:43:
            50:76:79:7e:a5:5a:75:af:19
        Exponent: 65537 (0x10001)
    X509v3 extensions:
        X509v3 Basic Constraints:
            CA:FALSE
        Netscape Cert Type:
            SSL Server
        X509v3 Key Usage:
            Digital Signature, Key Encipherment
        X509v3 Extended Key Usage:
            TLS Web Server Authentication, Netscape Server Gated Crypto,
                                        Microsoft Server Gated Crypto
        Netscape Comment:
            OpenSSL Certificate for SSL Web Server
Signature Algorithm: sha1WithRSAEncryption
    45:30:9d:04:0e:b7:86:9e:61:a1:b0:68:2b:44:93:1c:57:2a:
    99:42:bb:16:b1:ab:f5:c0:d2:33:12:c8:d3:1d:2b:bb:6b:9a:
    4c:c7:53:bc:e4:88:ef:1e:c3:37:ed:53:2c:15:cf:b8:90:df:
    df:4b:34:b8:db:cc:23:77:46:06:72:9d:43:60:a8:a2:ed:0a:
    bb:1a:a4:e8:4e:ba:66:93:63:74:87:fd:43:48:b6:93:a2:e3:
    3d:da:1b:64:46:35:88:b4:4b:22:e6:3c:84:70:5d:88:dd:64:
    c2:51:c2:d6:59:80:87:bc:bd:7f:e3:c1:45:7e:c0:5f:9c:ca:
    e1:a1
```

Chapter 3.

Implementation

3.1 Implementation

In this chapter is described the operation over the digital certificates as untangling digital certificates, obtaining of certificates, installing the certificates, the graphics, and the designing and implementation of the program with easy structure which allow to do some operation over the certificates in Windows operation system.

Untangling Digital Certificates

Digital certificates, more commonly just called certificates, are used by the SSL security protocol to encrypt, decrypt, and authenticate data. The certificate contains the owner's company name and other specific information that allows recipients of the certificate to identify the certificate's owner. The certificate also contains a public key used to encrypt the message being transported across the Internet. As I mentioned earlier, SSL uses two kinds of certificates: root certificates and server certificates. Root certificates are installed on the browser, and server certificates exist on the Web server. A root certificate tells the browser that you will accept certificates signed by the owner of the root certificate. For example, if you install a root certificate signed and issued by Blue Sand Software into your browser, you will be able to authenticate and decrypt messages that were sent from Blue Sand Software. This is vital to ensuring a secure transaction. Server certificate is installed on the Web server. It works much like the root certificate and is in charge of encrypting the messages sent to browsers and decrypting messages received from browsers.

So how do you get these certificates? You'll need to go to the Web site of one of the various certificate authorities and submit a Certificate Signing Request (CSR). After you submit the CSR, the certificate authority will verify that your business is valid and issue you a server certificate. You then install the server certificate on the Web server. But keep in mind that any user who wants to use your secure Web server must have the root certificate from the certificate authority installed on their browser as well. Most browsers have a Verisign root certificate preinstalled, so if Verisign is your server certificate authority, you're ready to go.

Breaking Down Encryption

SSL handles the scrambling of messages for you so that only the intended recipient can read it. The encryption/decryption process goes something like this:

1. The user browses to the secure Web server's site.
2. The user's SSL secured session is started and a unique public key is created for the browser (using the certificate authority's root certificate).
3. A message is encrypted and then sent from the browser using the server's public key. The message is scrambled during the transmission so that nobody who intercepts the message can make sense of it.
4. The message is received by the Web server and is decrypted using the server's private key.

The process of SSL encryption relies upon two keys: the server's public key and private key. The private key only exists on the Web server itself and is used by the Web server to encrypt and decrypt secure messages. The public key exists on any client computer that has installed a root certificate for that Web server. Once the public key is installed, the user can send encrypted messages to and decrypt messages received from the Web server. Figure 1 shows this process. Just to be extra safe, the keys are discarded once the transaction's session ends.

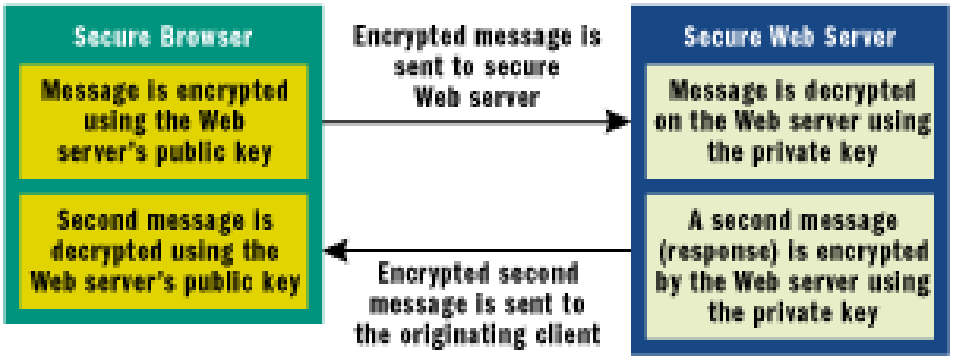


Figure 1 Asymmetrical Encription using SSL

SSL doesn't prevent the message from being intercepted. However, it does make the message useless to the rogue interceptor. In other words, someone could capture the message on its way to the secure Web server, but could not decrypt it because they do not have the server's private key.

The encryption process can be either symmetric or asymmetric. Symmetric encryption uses a single key by both parties to encrypt and decrypt secure messages. The problem is that the key itself has to be passed along as part of the conversation. This leaves the key, with its power to decrypt the messages,

easy prey to a seasoned cracker. Because of this downfall, asymmetric encryption was welcomed with open arms. Since the keys are never transported with asymmetric encryption, there is never a risk of them being stolen by a rogue third party. As you can see in Figure 1, using asymmetric encryption lets you tightly and securely transacts business via the Web.

With asymmetric encryption, why can't someone with the root certificate steal a secure message and decrypt it themselves? Well, each session key (or public key) generated by the root certificate is unique. So even if the cracker has the same root certificate installed on their browser, it will do them no good in their attempts at decrypting the message. In fact, even the originating browser of the encrypted message can't decrypt it. Only the Web server with the appropriate private key can decrypt the message.

3.2 Certificate authority

Certificates are signed by the Certificate Authority (CA) that issues them. In essence, a CA is a commonly trusted third party that is relied upon to verify the matching of public keys to identity, e-mail name, or other such information.

The benefits of certificates and CAs occur when two entities both trust the same CA. This allows them to learn each other's public key by exchanging certificates signed by that CA. Once they know each other's public key, they can use them to encrypt data and send it to one another, or to verify the signatures on documents.

A certificate shows that a public key stored in the certificate belongs to the subject of that certificate. A CA is responsible for verifying the identity of a requesting entity before issuing a certificate. The CA then signs the certificate using its private key, which is used to verify the certificate. A CA's public keys are distributed in software packages such as Web browsers and operating systems, or they can also be added manually by the user.

Software that is designed to take advantage of the PKI maintains a list of CAs that it trusts. In order to create a certificate, you need to create the base on which to tie your modules. This container, further referenced to as "Certificate envelope", is issued by a traditional The Certificate envelope is basically just like any other module, but since it is required for all other modules to function it was named certificate envelope. The required artifacts in each and every certificate envelope are:

• The certificate id is the artifact that uniquely identifies a certificate envelope and is required to map modules to a specific certificate envelope. Separating the identifier from the users public key creates a

more loosely coupled structure. By doing so, we gain traceability of a certificate, so that a validating source easily can identify and contact the certificate / revocation authority. In addition to this, it would be technically possible to reassign a certificate's key pair. Even though this would facilitate recovery from revocation, it would further complicate the already delicate problems of revocation distribution and checking.

• The public key of the certificate authority that issued the certificate envelope. This is the corresponding public key to the key used for signing the certificate envelope in order to ensure the integrity of the envelope.
• In order to tie the certificate to a person, a standard PKI keypair is used. In order for validating sources to verify that the party presenting the the certificate is the owner, a challenge is made using this public key as the cryptographic key.
• The certificate envelope is signed with the CA:s private key. The signature is used in conjunction with the CA:s public key in order to ensure the integrity of the certificate envelope.
• The date when the certificate was created.
• The date the certificate envelope is valid from.
• The date the certificate envelope is valid to.
• Optional, alternate way to look up possible certificate revocation. As an example, a
third-party revocation authority could be suggested here in case the primary authority is unreachable.

PKI sets up entities, called certificate authorities, that implement the PKI policy on certificates. The general idea is that a certificate authority is trusted, so that users can delegate the construction, issuance, acceptance, and revocation of certificates to the authority, much as one would use a trusted bouncer to allow only some people to enter a restricted nightclub. PKI leaves some room for interpretation; one of the areas where PKIs differentiates is the handling of certificates, which according to me also is the most crucial part. In "Security in Computing" the tasks a certificate authority should have are described as:
• managing public key certificates for their whole life cycle
• issuing certificates by binding a user's or system's identity to a public key with a digital signature
• scheduling expiration dates for certificates
• ensuring that certificates are revoked when necessary by publishing certificate revocation lists
Many PKI have a hierarchical structure for CAs and are built using a centralized architecture. However there are alternatives. PKI is categorized by the architecture types as follows: 1. Hierarchical PKI 2. Mesh PKI 3. Trust-file PKI

The difference is the way how they rely on CA (Certification Authority). A hierarchical PKI has the most significant CA in terms of trust at the root of the hierarchy tree. A mesh PKI has CAs issue cross-certificates to each other. A trust-file PKI has a local file of public-key certificates that the user trusts as starting points for certification chain

Unfortunately, we consider the requirement of signature validation less likely to be solved. Adding revocation checks for each signature in each module would require enormous amounts of traffic and administration. Even if the revocation systems and the validating clients were to withstand the traffic and computational power, the problem of signature administration for the users would still be a major problem. Every signing entity would have to keep a public record of revoked signatures. In effect, for every entity, there would have to be a signature repository. Requiring that every user constantly monitors the subject of a signature would not be feasible, and would make the system unusable for a major part of the intended users. On the other hand, supporting a signature revocation system that only a minority uses would instead lead to a false sense of security. While the trust system of PGP may work in small, well-known groups, we believe it would not work in a global identity system.

3.3 The operation over certificates.

This part will describe how action is handled by the different PKI architectures. These changing structures and methods will be compared and analyzed.

PKIX

There are two kinds of revokes in PKIX the first one is an automated one; when the lease time of a certificate have past, the certificate is automatically seen as revoked.

The other revocation case is when some authorized entity has requested a revocation of some certificate.

When revocation has been verified as the new certificate are put in a certificate revocation list, this list is later on published on the different repositories.

PGP

The weakest link of OpenPGP PKI is the revocation of public key. As there is no official channel for acquiring and distributing OpenPGP public keys, there are no guarantee about how to tell everyone that your key is no longer valid. The typical answer to this problem of PGP is to use PGP public key-server for distributing certification. “Typically, to communicate that a certificate has been revoked, a signed note, called a key revocation certificate, is posted on PGP certificate servers, and widely

distributed to people who have the key on their public key-rings. People wishing to communicate with the affected user, or use the affected key to authenticate other keys, are warned about the hazards of using that public key". However, there are few research on the PGP public key-server and usually the key-server is not considered as the part of OpenPGP PKI

PGP encryption uses public-key cryptography and includes a system which binds the public keys to a user name and/or an e-mail address. The first version of this system was generally known as a web of trust to contrast with the X.509 system which uses a hierarchical approach based on certificate authority and which was added to PGP implementations later. Current versions of PGP encryption include both options through an automated key management server. In the (more recent) OpenPGP specification, trust signatures can be used to support creation of certificate authorities. A trust signature indicates both that the key belongs to its claimed owner and that the owner of the key is trustworthy to sign other keys at one level below their own. A level 0 signature is comparable to a web of trust signature since only the validity of the key is certified. A level 1 signature is similar to the trust one has in a certificate authority because a key signed to level 1 is able to issue an unlimited number of level 0 signatures. A level 2 signature is highly analogous to the trust assumption users must rely on whenever they use the default certificate authority list (like those included in web browsers); it allows the owner of the key to make other keys certificate authorities.

PGP versions have always included a way to revoke identity certificates. A lost or compromised private key will require this if communication security is to be retained by that user. This is, more or less, equivalent to the certificate revocation lists of centralized PKI schemes. Recent PGP versions have also supported certificate expiration dates.

The problem of correctly identifying a public key as belonging to a particular user is not unique to PGP. All public key / private key cryptosystems have the same problem, if in slightly different guise, and no fully satisfactory solution is known. PGP's original scheme, at least, leaves the decision whether or not to use its endorsement/vetting system to the user, while most other PKI schemes do not, requiring instead that every certificate attested to by a central certificate authority be accepted as correct.

PGP, on the other hand, uses a decentralized certification system called .The Web of Trust.. The system is self-certifying, people who already trust each other, sign each others keys. In this way small rings of trusted groups build up, and start to join forming a larger web. Someone who is not part of the web, or is part of an unconnected subsection of the web, can still communicate with other people, they just have to take it on trust that they are who they say they are. This is not a huge risk when you consider the concept of identity on the Internet. If you are mailing someone you have never met

before, in response to a post to a newsgroup for instance, then you do not actually need to know who they are; you are only responding to what they have written. On the other hand if you are passing confidential data to a counterpart in another company, that you have never met before, then you would naturally ring them or meat them in person. Once you actually trust the person, PGP employs a fingerprint method (hash of the public key), to check that the certificate you have got is actually that of the person you met or rang.

PGP, on the other hand, uses a decentralized certification system called .The Web of Trust. The system is self-certifying, people who already trust each other, sign each others keys. In this way small rings of trusted groups build up, and start to join forming a larger web. Someone who is not part of the web, or is part of an unconnected subsection of the web, can still communicate with other people, they just have to take it on trust that they are who they say they are. This is not a huge risk when you consider the concept of identity on the

Internet. If you are mailing someone you have never met before, in response to a post to a newsgroup for instance, then you do not actually need to know who they are; you are only responding to what they have written. On the other hand if you are passing confidential data to a counterpart in another company, that you have never met before, then you would naturally ring them or meat them in person. Once you actually trust the person, PGP employs a fingerprint method (hash of the public key), to check that the certificate you have got is actually that of the person you met or rang.

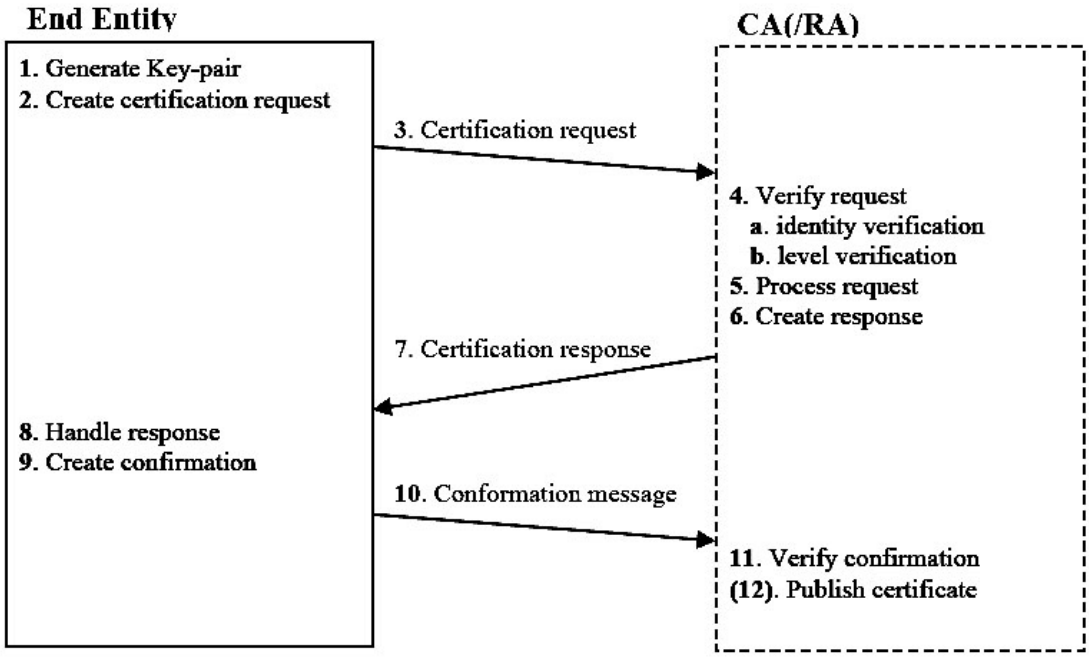
Suggested solution

The X.509 framework defines the CA to distribute CRLs for end entities to use as a revocation reference. In the proposed solution, validation is made by the CA it self, this makes revocation faster; the revocation requests are validated and carried out, since each validation goes through the CA the revocation becomes instant. The suggested solution is based on the X.509 framework. This framework has good structure use centralized certificate authority

First registration

First registration/certification: This is the process where an end entity first makes itself known to a CA or RA, prior to the CA issuing a certificate or certificates for that end entity. The end result of this process (when it is successful) is that a CA issues a certificate for an end entity's public key, and returns that certificate to the end entity and/or posts that certificate in a public repository.

This scheme will follow that of the original scheme in X.509 (a three way handshake), with some modifications.

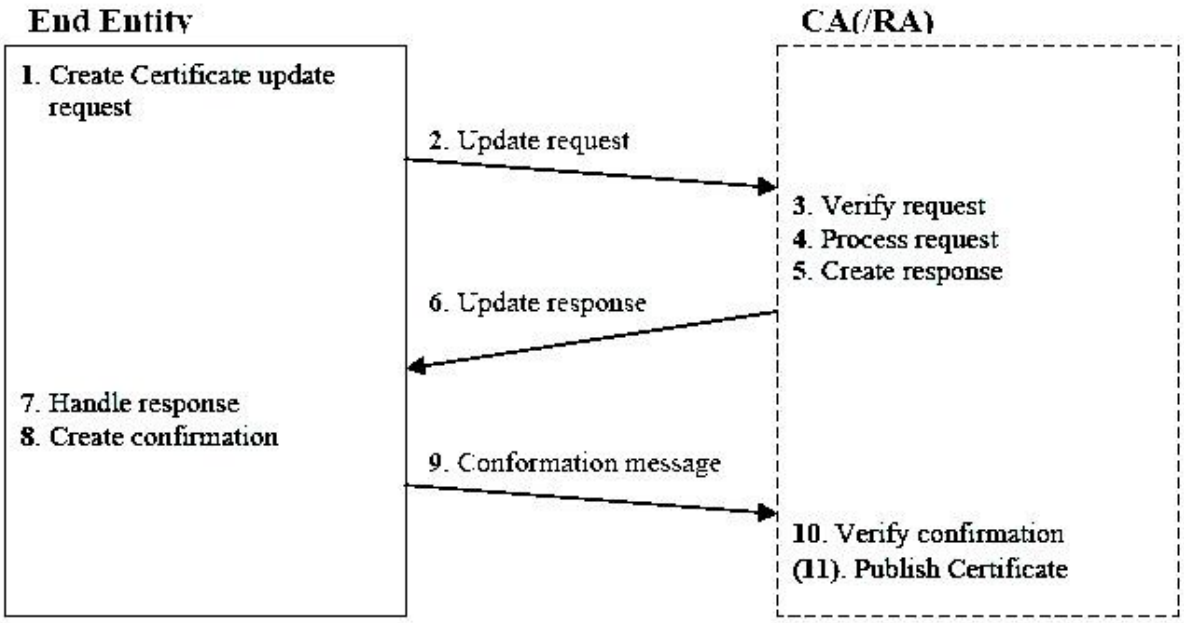


Initial registration scheme

In "Handle response" (step 8) a task is added at the end entity; to process the information regarding the level appointed. If some part of the handshake (step 3, 7 and 10) fails, the certificate authority automatically revocation the newly created certificate as in X.509. To add some level of security to this scheme, I propose that the CA puts the created by not yet confirmed certificates in a "to be verified" structure, before adding it to the lists of certificates, in this way it doesn't have to be revoked (it have never been verified). These certificates will then upon confirmation be added to the list and the validation process. This would be the last step in the, initial registration scheme and followed by publication of certificate

Certificate update

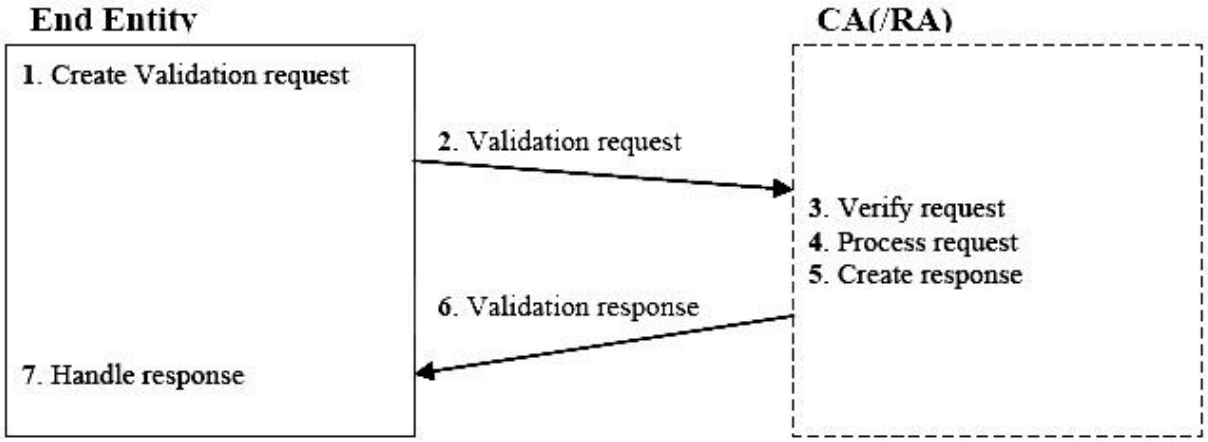
Certificate update: As certificates expire they may be "refreshed" if nothing relevant in the environment has changed. As the definition above implies this scheme is only used when the environment is secure, therefore not used in the higher certificate levels of security. This is an important part of the trust issue that the proposed solution build on; don't even take the risk of getting compromised.



Certificate update scheme

As in Key-pair update a proof of possession, verifies the entity in step 3. After verification the update process involves “repackaging”; the old key is reused, but some attributes need to be updated. When repackaging is done, a new one-way-hash has to be computed and updated in the validation database along with the new lease time and the certificate. In case of failure in any of the three parts of the handshake (step 2, 6 and 9), both, new and old, certificates will automatically be annulated.

Certification validation

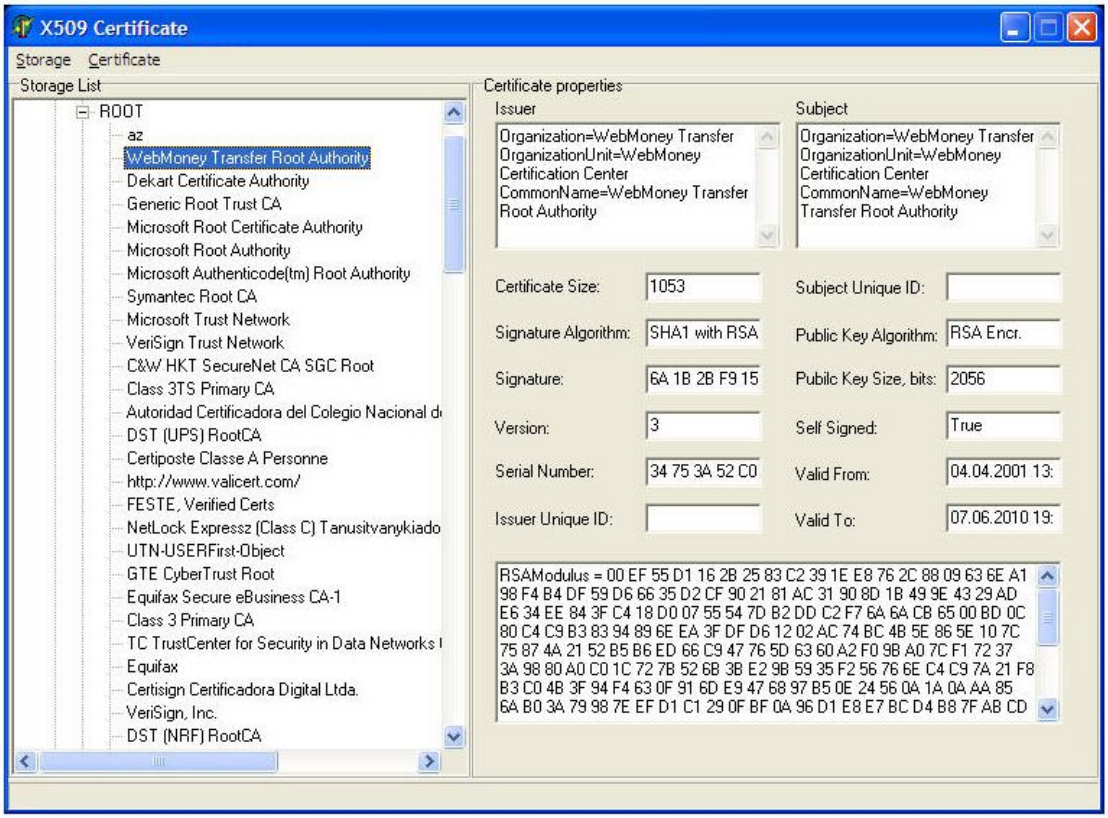


To validate a downloaded certificate, the end entity constructs a validation request message (step 1) containing (amongst other attributes) a one-way-hashed value of the certificate in question. After verifying the requesting entity the one-way-hash value is used for validating the certificate (step 4), if the value matches a value in the validation database, it is validated else revoked. The validation response (Figure 8; step 6) contains the outcome of the validation. If the verification succeeds, communication can be initiated (step 7). The advantage using this certificate structure, in comparison to writing a new structure, is that no changes have to be to the "parsers" of existing architectures. Parsing and handling the security levels are as explained earlier an implementation issue.

3.4 Designing of program.

In Windows operating system there are no regular techniques for working with certificates and the given program allows to work effectively with storehouse of certificates, to look through, add, delete, and also to export and to import from storehouse. The storehouse of certificates is used by all programs of Windows operating system for example Outlook. To transfer certificates from one computer to another it is possible but it difficult and long to do.

In the screen of program you can see the list of certificates. In this case we can see the properties of Web money Transfer Root Certificate, which is stored in storehouse of Window, the certificate size, signature algorithm, public key algorithm, version, serial number of certificate, validation of certificate.


X509 Certificate
Storage
Certificate
Storage List
ROOT
az
WebMoney Transfer Root Authority
Dekart Certificate Authority
Generic Root Trust CA
Microsoft Root Certificate Authority
Microsoft Root Authority
Microsoft Authenticode(tm) Root Authority
Symantec Root CA
Microsoft Trust Network
VeriSign Trust Network
C&W HKT SecureNet CA SGC Root
Class 3TS Primary CA
Autoridad Certificadora del Colegio Nacional d
DST (UPS) RootCA
Certiposte Classe A Personne
http://www.valicert.com/
FESTE, Verified Certs
NetLock Expressz (Class C) Tanusitvanykiado
UTN-USERFirst-Object
GTE CyberTrust Root
Equifax Secure eBusiness CA-1
Class 3 Primary CA
TC TrustCenter for Security in Data Networks
Equifax
Certisign Certificadora Digital Ltda.
VeriSign, Inc.
DST (NRF) RootCA
Certificate properties
Issuer
Organization=WebMoney Transfer
OrganizationUnit=WebMoney
Certification Center
CommonName=WebMoney Transfer
Root Authority
Subject
Organization=WebMoney Transfer
OrganizationUnit=WebMoney
Certification Center
CommonName=WebMoney
Transfer Root Authority
Certificate Size:
1053
Subject Unique ID:
Signature Algorithm:
SHA1 with RSA
Public Key Algorithm:
RSA Encr.
Signature:
6A 1B 2B F9 15
Pubilc Key Size, bits:
2056
Version:
3
Self Signed:
True
Serial Number:
34 75 3A 52 C0
Valid From:
04.04.2001 13:
Issuer Unique ID:
Valid To:
07.06.2010 19:
RSAModulus = 00 EF 55 D1 16 2B 25 83 C2 39 1E E8 76 2C 88 09 63 6E A1
98 F4 B4 DF 59 D6 66 35 D2 CF 90 21 81 AC 31 90 8D 1B 49 9E 43 29 AD
E6 34 EE 84 3F C4 18 D0 07 55 54 7D B2 DD C2 F7 6A 6A CB 65 00 BD 0C
80 C4 C9 B3 83 94 89 6E EA 3F DF D6 12 02 AC 74 BC 4B 5E 86 5E 10 7C
75 87 4A 21 52 B5 B6 ED 66 C9 47 76 5D 63 60 A2 F0 9B A0 7C F1 72 37
3A 98 80 A0 C0 1C 72 7B 52 6B 3B E2 9B 59 35 F2 56 76 6E C4 C9 7A 21 F8
B3 C0 4B 3F 94 F4 63 0F 91 6D E9 47 68 97 B5 0E 24 56 0A 1A 0A AA 85
6A B0 3A 79 98 7E EF D1 C1 29 0F BF 0A 96 D1 E8 E7 BC D4 B8 7F AB CD

Conclusion

The very first question a Web server administrator must confront is, "Do I really want/need to provide dynamic content from my server?" While dynamic content has allowed for a diverse range of user interaction and become a de facto standard for most large Web sites, it remains one of the largest security threats on the Internet. CGI applications and SSI-enabled pages are not inherently insecure, but poorly written code can potentially open up dangerous back doors and gaping holes on what would otherwise be a well-secured system.
The following are the three most common security risks CGI applications and SSI pages create:

Information leaks: Providing any kind of system information to a hacker could potentially provide a hacker with the ammunition they need to break into your server. The less a hacker knows about the configuration of a system, the harder it is to break into.

Access to potentially dangerous system commands/applications: One of the most common exploits used by hackers is to "take over" a service running on the server and use it for their own purposes. For example, gaining access to a mail application via an HTML form-based script, and then harnessing the mail server to send out spam or acquire confidential user information.

Depleting system resources: While not a direct security threat per se, a poorly written CGI application can use up a system's available resources to the point where it becomes almost completely unresponsive.

For security purposes administrators are responsible to: keep current with security patches, evaluate and expeditiously apply as appropriate; keep the operating system at a level supported by the vendor; properly restrict access to information; ensure the administrator of the server, or a designate, will be available during working hours to resolve problems; keep and make available a current list of phone numbers where administrators or designates may be reached in a critical situation outside normal hours.

The intent was to create a model that improved the way revocation is handled in terms of:
Revocation time: By constructing a new validation scheme, that puts the certificate authority in charge, the trust mechanism is further centralized (less entities involved in decisions). This new scheme minimizes the time it takes for all entities to be aware of the revoked certificates, whereas the Revocation Time requirement is fulfilled.

Security: When using the new schemes, all communicating entities contacts certificate authority and requests validation of the entities they which to communicate with. Since revocation time has been reduced to nearly non-existing, security regarding revocation has been enhanced, whereas the Security requirement has been fulfilled.
Scalability: Different levels of security have been nested into the original (X.509) certificate and structures have been created to ensure that the levels are followed. Different environments could use different levels of security, whereas the Scalability requirement has been fulfilled. The requirements have been fulfilled with small changes done to the original (X.509) framework. The reuse of existing certificate structures and communication protocols is for the better; fewer changes have to be made to existing architectures, which makes it doable.

For servers containing mission critical information we recommend also: provide high availability; provide users with the ability to audit use via logs and monitor exceptions; provide backup of data provide back out procedures for installations of new software or configurations; upgrade (server and application software and hardware) and provide capacity planning; regularly attend the WWW-Tech meetings and provide updates on progress and problems as well as new software functions available.

References

1. RSA Cryptography Standard, http://www.rsa.com/rsalabs/node.asp?id=2125 20 March 2008.

2. G. Bell, D. Siewiorek and S.H. Fuller “A ;ew Solution of Dijkstra's Concurrent Programming Problem”, Communication for ACM 17(8)

3. P. K. Yuen “Practical cryptology and Web security”

4. An Introduction to Cryptography. Network Associates, Inc., h t tp://www.pgpi.org/doc/pgpintro

5. The SSL Protocol, version 3.0. Netscape, Inc., h t tp://home.netscape.com/eng/ssl3

6. C. Kaufman, R. Perlman, and M. Speciner. Network Security: Private Communication in a Public World. Prentice Hall PTR

7. RSA Security. h t tp://www.rsa.co m .

8. R. L. Rivest and et al. The RC6 Block Cipher. RSA Security, h t tp://csrc.nist.gov/encryption/

9. P. Fergguson and G. Huston. What is a VPN. h t tp://www.employees.org/ferguson

10. CryptSoft Technologies. htttp://www.cryptsoft.com

11. Advanced EncryptionStandard (AES) Development Effort. US Government, h t tp://csrc.nist.gov

12. C. Burnwick and et al. The Mars Encryption Algorithm. IBM, h t tp://csrc.nist.gov/encryption/aes

13. http://en.wikipedia.org/wiki/Ssl

14. http://support.microsoft.com/

15. http://blogs.msdn.com/ie/archive/2005/04/20/410240.aspx

16. http://www.ibdarb.ru/